Alibaba Group 阿里巴巴集团 | 阿里技术

九年双11 互联网技术超级工程

赠阅

VISIT…

LANZAROTE Caliente.com

阿里技术

扫一扫二维码图案，关注我吧

「阿里技术」微信公众号

「阿里技术」官方微博

Alibaba Group 阿里巴巴集团 | 阿里技术

九年双11 互联网技术超级工程

序

阿里巴巴 CTO 行癫：
阿里双 11 是世界互联网技术的超级工程

2017 年 11 月 11 日晚上 10 点，阿里巴巴集团 CTO 张建锋（花名：行癫）连线上海双 11 媒体中心，为 700 多位中外媒体记者介绍了阿里技术在双 11 中取得的突破与成绩，表示“阿里双 11 是名副其实的世界互联网技术超级工程。”

以下为演讲全文：

阿里的双 11 是一次全球商业、科技、数据、智能的大协同，是一个商业社会的大协同，更是一个技术的大协同，是名副其实的世界互联网技术的超级工程。

今年的双 11，阿里技术能力再创新记录，达到了每秒 32.5 万笔的交

阿里巴巴集团 CTO & 双 11 技术指挥官 行癫

易创建峰值和每秒 25.6 万笔的支付成功峰值。在我们采用了非常多样的促销、营销手段的情况下，阿里的技术经受住了考验，给消费者创造了良好的体验。

今年的双 11，我们不仅是做了一次系统保障，也是技术全面赋能双 11。而且通过这个技术，也给我们的合作伙伴，包括参与双 11 的金融、物流、保险等等，也提供了全方位的支撑。

今年，数据智能、机器智能已经融入到整个系统的每一个方面：整个商品推荐、整个交易链路、包括整个决策都是用机器智能来做的：鲁班智能设计系统双 11 期间产生超过 4 亿的海报；智能客服系统承担了 95% 的客服咨询量；推荐系统每分每秒都通过用户的点击，记录他们的行为并分析、处理，然后来做决策；系统保障部分，包括我们的机房、巡检、压力测试，大规模用了机器智能技术，是人跟机器大规模的协同；智能音箱天猫精灵在双 11 期间销售超 100 万台，更是让阿里巴巴的机器智能以消费产品的形式进入百万家庭。

未来，我们希望通过阿里云平台，更好的支撑我们每一个商家、合作伙伴和客户——让他们通过阿里云平台能够第一时间享受到阿里这么多年积累下来的技术，让他们能够利用当今最好的技术更好地服务自己的消费者和客户。我们希望将技术变成全社会能够共享的普惠基础设施，从而降低整个社会的创新成本，提升整个社会的创新效率。

鸣谢

感谢“阿里巴巴技术大学”平台对本书的大力支持。“阿里巴巴技术大学”是阿里巴巴面向全体技术人的学习摇篮，秉承着“传承”“突破”“开拓”的办学理念，融汇阿里技术文化与技术实践，促动技术人才能力和视野提升。

感谢无私分享的阿里技术人，是你们才有这本极具价值的书。此书囊括详实的双 11 技术沉淀、业界前沿的技术突破。在此谨向作者的薪火相传，表达最深挚的感激。

最后，深深感谢阅读此书的你。感谢你在繁忙的工作中，抽出宝贵的时间，与我们共同携手走在技术进步的道路上。

谨在此向诸位致以最诚挚的谢意。2018，我们一起遇见更好的自己。

阿里巴巴技术大学现场照

目录

新智能 1

阿里搜索技术，在 AI 路上走了多远? 1

直击阿里新一代数据库技术：如何实现极致弹性能力? 9

争分夺秒：阿里实时大数据技术全力助战双 11 21

阿里小蜜这一年：从点到面的变迁 41

菜鸟仓配自动化 UCS 揭秘 62

阿里怎么发红包? 自研智能权益系统首次公开 68

2017 双 11：开启智能全链路压测之路 78

智能写手：智能文本生成在 2017 双 11 的应用 85

浅谈分布式存储系统 Pangu2.0：它让双 11 运维变得智能起来 100

新基础 118

双 11 稳定性负责人叔同讲述：九年双 11 的云化架构演进和升级 118

阿里双 11 网络技术揭秘：百万级物理和虚拟网络设备的智能化之路 134

从 10% 到 40%：阿里巴巴混部技术权威详解 150

经历 400 多天打磨，HSF 的架构和性能有哪些新突破? 166

直击阿里容器技术 Pouch 179

直击阿里双 11 神秘技术：PB 级大规模文件分发系统“蜻蜓” 185

双 11 万亿流量下的分布式缓存 203

2017 双 11 交易系统 TMF2.0 技术揭秘，实现全链路管理 214

新体验 221

一天造出 10 亿个淘宝首页，阿里工程师如何实现? 221

双十一安全技术：目标检测在淘宝直播中的应用 238

持续迭代下的双 11 供应链体系最新架构及功能解读 244

七层流量清洗提供安全防护新方案 252

2017 双 11：区块链在天猫国际商品溯源中的应用 260

直击 Weex 在优酷双 11 猫晚直播的应用 270

如何把范冰冰“送”到你家? 双 11 晚会“逆天”技术首次公开 281

阿里搜索技术，在 AI 路上走了多远?

三桐

阿里妹导读：以深度学习为代表的人工智能在图像、语音和 NLP 领域带来了突破性的进展，在信息检索和个性化领域近几年也有不少公开文献，比如 wide& deep 实现了深度模型和浅层模型的结合，dssm 用于计算语义相关性，deepfm 增加了特征组合的能力，deep CF 用深度学习实现协同过滤，rnn recommender 采用行为序列预估实现个性化推荐等。

工业级的信息检索或个性化系统是一个复杂的系统工程，深度学习的工业级应用需要具备三个条件：强大的系统计算能力，优秀的模型设计能力和合适的应用场景。今天，我们邀请了阿里搜索事业部资深算法专家三桐，介绍阿里在深度学习系统、深度学习算法和搜索应用落地的进展和思考，希望对大家有所启发。

深度学习在搜索的应用概括起来包括 4 个方面：

首先是系统：强大的深度学习训练平台和在线预测系统是深度学习应用的必要条件，目前我们的离线深度学习框架、在线深度学习框架和在线预测框架统一到 tf，并实现了日志处理，特征抽取，模型训练和在线服务部署端到端的流程，极大提升了算法迭代效率；

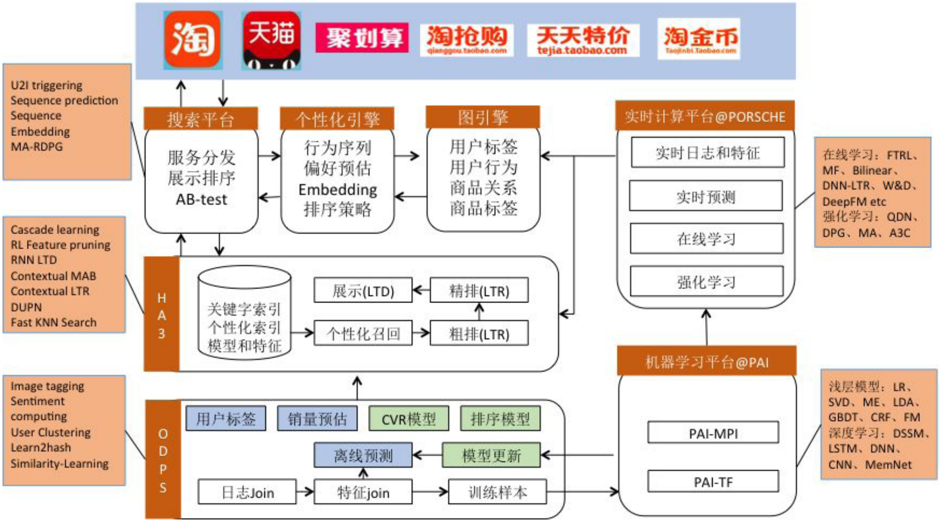
其次是搜索应用：包括智能交互，语义搜索，智能匹配和智能决策四个技术方向，这四个方向的协同创新实现了搜索全链路的深度学习技术升级，并具备从传统的单场景单目标优化到多场景多目标联合优化的能力；

再次是在性能优化上做的工作：包括模型压缩、量化、低秩分解再到二值网络，大量的技术调研和论证，为未来提高深度模型预测性能和软硬件协同优化做了很好的

技术铺垫；

最后是排序平台化：实现了 PC 商品搜索、无线商品搜索、店铺内搜索搜索和店铺搜索的搜索服务统一，通过特征和模型复用，实现了多条业务线技术的快速升级。下面我会简要的概括下在四个方向上取得的主要进展和背后的思考。

下面是搜索系统和算法的简图。系统包括：



搜索系统和算法简图

a. 离线数据平台 ODPS，负责离线日志 join、特征抽取和离线模型预估产出排序特征，时效性不强的特征都是通过离线数据平台产出的，比如用户性别标签，商品关键字等；

b. 离线机器学习平台 PAI，底层是主流的 parameter server 和 TF 深度学习框架，平台实现了大部分机器学习算法模型的并行训练和预测，在搜索应用中主要作用是离线模型训练产出离线排序特征模型；

c. 流式计算和在线学习平台 Porsche，流式计算是基于 blink 负责实时日志解析和特征 join 生成实时排序特征，在线学习和离线学习底层框架可以相同，差别主要是依赖数据源和部分优化方法不同，由于用户行为和市场环境变化快，

流式计算和在线学习在搜索应用非常广泛，并积累了不少在线学习和强化学习算法；

d. 在线服务平台，包括引擎、排序服务和搜索平台组成，负责在线的服务分发、索引查询、排序服务和结果合并等功能，搜索的排序策略、相关性、个性化等模型主要通过在线预测服务生效。经过多年发展我们已经具备了非常完善的商品搜索排序算法体系，包括知识图谱、分词、tagging、类目预测、意图预测、拼写纠错、query 推荐、query 语义改写、相关性、商品标签、商品质量、店铺分层、用户 profile、用户偏好、用户感知、召回策略、个性化模型、多样性策略、异构服务混排策略、多目标联合优化策略、多场景联合排序策略等，并平台化的方式赋能相关业务团队。

系统进展：机器学习平台和在线预测平台

机器学习平台。搜索训练样本主要来自用户行为，由于用户行为是流式数据，适合做在线深度学习，但当模型参数非常庞大需要海量的样本时在线学习需要很长的时间才能收敛，这时一般是先做离线预训练再结合增量或在线学习，另外有些模型离线预训练后在线只需要对接近输出层的网络做 fine-tuning。搜索在实际应用的有离线机器学习平台 PAI 和在线机器学习平台 Porsche，两个平台深度学习框架目前都统一到了 tf-pai，tf-pai 对原生 tf 做了一些优化，比如底层通讯，稀疏参数存储、优化方法、GPU 显存优化等，比原生 tf 训练深度有较大的提升，训练上千亿样本和上百亿参数的深度模型毫无压力。

虽然 Porsche 和 PAI 都支持 GPU，但在搜索应用中 CPU 依然是主流，GPU 应用比较少，原因主要是个性化相对图像或语音简单，特征抽取网络比较浅，维度相对较低，GPU 的稠密矩阵计算能力得不到充分发挥，同时离在线混布后流量低谷期间腾出了大量的在线服务闲置 CPU，把临时闲置的 CPU 利用起来做深度学习训练是一个非常好的思路。

在线预估 RTP，搜索排序算分服务。由于每次搜索请求有上千个商品需要计算排序分数，深度模型应用对 RTP 服务的压力是非常大的，RTP 通过采用异构计算，

计算算子化和模型分片等方式解决了深度模型 inference 计算和存储问题，深度模型用 GPU，浅层模型用 CPU，今年双 11 期间搜索 RTP 服务用到了 550 张 GPU 卡。另外，RTP 还实现了离线 / 在线训练模型 / 数据和在线预测服务部署的无缝衔接，算法训练好的模型或数据可以很轻松的部署都在线服务，提升了算法迭代效率。

算法：智能交互、语义搜索、智能匹配和搜索策略四个方向

智能交互。商品搜索就是带交互的商品推荐，用户通过关键字输入搜索意图，引擎返回和搜索意图匹配的个性化推荐结果，好的交互技术能够帮助到用户更好的使用搜索引擎，目前搜索的交互主要是主动关键字输入和关键字推荐，比如搜索框中的默认查询词和搜索结果中的文字链等，推荐引擎根据用户搜索历史、上下文、行为和状态推荐关键字。

和商品推荐的区别是，关键字推荐是搜索链路的中间环节，关键字推荐的收益除了关键字的点击行为外，还需要考虑对整个购物链路的影响，包括在推荐关键字的后续行为中是否有商品点击、加购和成交或跳转到另外一个关键字的后继行为，这是一个典型的强化学习问题，action 是推荐的关键字候选集合，状态是用户当前搜索关键词、上下文等，收益是搜索引导的成交。

除了被动的关键字推荐，我们也在思考搜索中更加主动的交互方式，能够做到像导购员一样的双向互动，主动询问用户需求，挑选个性化的商品和给出个性化的推荐理由，目前阿里搜索团队已经在做智能导购和智能内容方向的技术原型及论证，智能导购在技术上主要是借鉴对话系统，通过引导用户和引擎对话与关键字推荐方式互为补充，包括自然语言理解，对话策略，对话生成，知识推理、知识问答和商品搜索等模块，功能主要包括：

a. 根据用户搜索上下文生成引导用户主动交互的文本，比如搜索“奶粉”时，会生成“您宝宝多大？ 0~6 个月，6 个月到 1 岁….”引导文案，提示用户细化搜索意图，如果用户输入“3 个月”后，会召回相应段位的奶粉，并在后续的搜索中会记住对话状态“3 个月”宝宝和提示用户“以下是适合 3 个月宝宝的奶粉”。

b. 知识导购，包含提高售前知识问答或知识提示，比如“3 个月宝宝吃什么奶粉”回答“1 段”。目前对话技术正在提高中，尤其是在多轮对话状态跟踪、知识问答和自动评价几个方面，但随着深度学习、强化学习和生成对抗学习等技术在 NLP、对话策略、阅读理解等领域的应用，越来越多的训练数据和应用场景，domainspecific 的对话技术未来几年应该会突飞猛进。智能内容生成，包括生成或辅助人工生成商品和清单的“卖点”，短标题和文本摘要等，让淘宝商品表达更加个性化和多元化。

语义搜索。语义搜索主要是解决关键字和商品内容之间的语义鸿沟，比如搜索“2~3 周岁宝宝外套”，如果按照关键字匹配召回结果会远小于实际语义匹配的商品。

语义搜索的范围主要包括：

a. query tagging 和改写，比如新品，年龄，尺码，店铺名，属性，类目等搜索意图识别和归一化，query tagging 模型是用的经典的序列标注模型 bi-lstm + CRF，而标签分类（归一化）作为模型另外一个任务，将序列标注和分类融合在一起学习。

b. query 改写，主要是计算 query 之间相似度，把一个 query 改写成多个语义相似的 query，通常做法是先用不同改写策略生成改写候选 query 集合，比如词替换、向量化后 top k、点击商品相似度等，然后在用 ltr 对后续集合排序找出合适的改写集合，模型设计和训练相对简单，比较难的是如何构建高质量的训练样本集合，线上我们用 bandit 的方法探测部分 query 改写结果的优劣，离线则用规则和生成对抗网络生成一批质量较高的样本。

c. 商品内容理解和语义标签，通过商品图片，详情页，评价和同义词，上下位词等给商品打标签或扩充商品索引内容，比如用 image tagging 技术生成图片的文本标签丰富商品内容，或者更进一步用直接用图片向量和文本向量融合，实现富媒体的检索和查询。

d. 语义匹配，经典的 DSSM 模型技术把 query 和商品变成向量，用向量内积表达语义相似度，在问答或阅读理解中大量用到多层 LSTM + attention 做语义

匹配，同样高质量样本，特别是高质量负样本很大程度上决定了模型的质量，我们没有采样效率很低的随机负采样，而是基于电商知识图谱，通过生成字面相似但不相关的 query 及相关文档的方法生成负样本。

从上面可以看到 querytagging、query 相似度、语义匹配和语义相关性是多个目标不同但关联程度非常高的任务。下一步计划用统一的语义计算框架支持不同的语义计算任务，具体包括：

1. 开发基于商品内容的商品表征学习框架，为商品内容理解，内容生成，商品召回和相关性提供统一的商品表征学习框架，重点包括商品标题，属性，详情页和评价等文本信息抽取，图像特征抽取和多模信号融合。
2. query 表征学习框架，为 query 类目预测，query 改写，query 推荐等提供统一的表征学习框架，重点通过多个 query 相似任务训练统一的 query 表征学习模型。
3. 语义召回，语义相关性等业务应用模型框架。语义搜索除了增加搜索结果相关性，提升用户体验外，也可以一定程度上遏制淘宝商品标题堆砌热门关键词的问题。

智能匹配。这里主要是指个性化和排序。内容包括：

a. ibrain(深度用户感知网络)，搜索或推荐中个性化的重点是用户的理解与表达，基于淘宝的用户画像静态特征和用户行为动态特征，我们基于 multi-modalslearning、multi-taskrepresentation learning 以及 LSTM 的相关技术，从海量用户行为日志中直接学习用户的通用表达，该学习方法善于“总结经验”、“触类旁通”，使得到的用户表达更基础且更全面，能够直接用于用户行为识别、偏好预估、个性化召回、个性化排序等任务，在搜索、推荐和广告等个性化业务中有广泛的应用场景，感知网络超过 10B 个参数，已经学习了几千亿次的用户行为，并且会保持不间断的增量学习越来越聪明。

b. 多模学习，淘宝商品有文本、图像、标签、id 、品牌、类目、店铺及统计特

征，这些特征彼此有一定程度的冗余和互补，我们利用多模学习通过多模联合学习方法把多维度特征融合在一起形成统一的商品标准，并多模联合学习中引入 self-attention 实现特征维度在不同场景下的差异，比如女装下图片特征比较重要，3C 下文本比较重要等。

c. deepfm，相对 wide& deep 模型，deepfm 增加了特征组合能力，基于先验知识的组合特征能够应用到深度学习模型中，提升模型预测精度。

d. 在线深度排序模型，由于行为类型和商品重要性差异，每个样本学习权重不同，通过样本池对大权重样本重复 copy 分批学习，有效的提升了模型学习稳定性，同时通过融合用户状态深度 ltr 模型实现了千人千面的排序模型学习。

e. 全局排序，ltr 只对单个文档打分然后按照 ltr 分数和打散规则排序，容易导致搜索结果同质化，影响总页效率，全局排序通过已知排序结果做为上下文预测下一个位置的商品点击概率，有效提升了总页排序效率。

f. 另外工程还实现了基于用户和商品向量的向量召回引擎，相对倒排索引，向量化召回泛化能力更强，对语义搜索和提高个性化匹配深度是非常有价值的。

以上实现了搜索从召回、排序特征、排序模型、个性化和重排的深度学习升级，在双 11 无线商品搜索中带来超过 10% (AB-Test) 的搜索指标提升。

多智能体协同学习实现智能决策

搜索中个性化产品都是成交最大化，导致的问题是搜索结果趋同，浪费曝光，今年做的一个重要工作是利用多智能体协同学习技术，实现了搜索多个异构场景间的环境感知、场景通信、单独决策和联合学习，实现联合收益最大化，而不是此消彼长，在今年双 11 中联合优化版本带来的店铺内和无线搜索综合指标提升 12% (AB-Test)，比非联合优化版本高 3% (AB-Test)。

性能优化。在深度学习刚起步的时候，我们意识到深度模型 inference 性能会是一个瓶颈，所以在这方面做了大量的调研和实验，包括模型压缩 (剪枝)，低秩分解，量化和二值网络。

通过以上技术，今年双 11 期间在手淘默认搜索、店铺内搜索、店铺搜索等均取

得了 10% (AB-Test) 以上的搜索指标提升。

阿里巴巴人工智能搜索应用的未来计划

通用用户表征学习

前面介绍的 DUPN 是一个非常不错的用户表征学习模型，但基于 query 的 attention 只适合搜索，同时缺少基于日志来源的 attention，难以推广到其他业务，在思考做一个能够适合多个业务场景的用户表征模型，非搜索业务做些简单 fine tuning 就能取得比较好的效果；同时用户购物偏好受季节和周期等影响，时间跨度非常大，最近 K 个行为序列假设太简单，我们在思考能够做 life-long learning 的模型，能够学习用户过去几年的行为序列；

搜索链路联合优化

从用户进入搜索到离开搜索链路中的整体优化，比如搜索前的 query 引导（底纹），搜索中的商品和内容排序，搜索后的 query 推荐（锦囊）等场景；

跨场景联合优化

今年搜索内部主搜索和店铺内搜索联合优化取得了很好的结果，未来希望能够拓展在更多大流量场景，提高手淘的整体购物体验；

多目标联合优化

搜索除了成交外，还需要承担卖家多样性，流量公平性，流量商业化等居多平台和卖家的诉求，搜索产品中除了商品搜索外还有“穹顶”，“主题搜索”，“锦囊”，“内容搜索”等非商品搜索内容，不同搜索目标和不同内容（物种）之间的联合优化未来很值得深挖。

智能交互

“搜索排序做的再好，搜索也只是一个工具”，如何把搜索从工具做成私人导购助手，能够听懂你的语言，了解你的情绪，能够对话和多轮交互，解决售前售后困惑，推荐更加个性化的商品应该是搜索未来的另外一个主要发展方向。

直击阿里新一代数据库技术：如何实现极致弹性能力？

张瑞

阿里妹导读：张瑞，阿里巴巴研究员，阿里集团数据库技术团队负责人，经历阿里数据库技术变革历程，连续六年作为数据库总负责人参与双 11 备战工作。今天，我们邀请他来分享新一代数据库技术在双 11 中的应用。

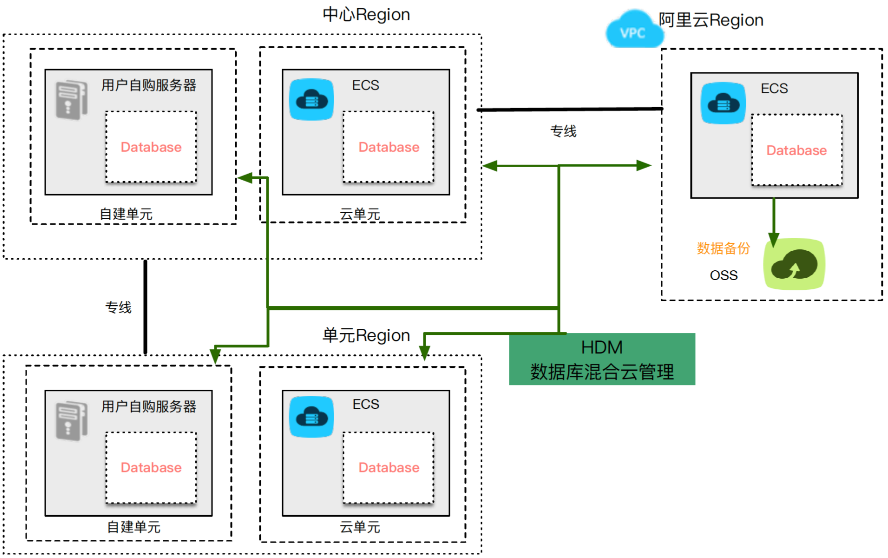
阿里数据库技术团队负责人张瑞

张瑞：双 11 是一场技术大练兵，是互联网界的超级工程。需要做到支撑尽可能高的零点峰值，给用户最好的体验；也要做到成本尽可能低，要求极致的弹性能力；还要做到整体系统的稳定。

数据库如何实现极致弹性能力?

数据库上云

大家都知道，数据库实现弹性能力是比较困难的，一方面是因为数据库对性能要求非常高，另一方面是需要进行大量数据的搬迁，成本很高。数据库弹性的第一个方向是数据库上云，通过云的弹性能力来解决数据库的资源问题。



数据库上云面临以下几个难点：

1. 数据库如何快速上云，构建混合云?
2. 如何降低虚拟化带来的性能损耗?
3. 公有云环境和内部网络的互通问题。

经过几年的探索，这些难点都已得到解决。第一，数据库使用了高性能 ECS，通过使用 SPDK、DPDK 技术和 NVMe 存储，可以让虚拟化损耗非常小，接近物理机；第二，我们建设了一套数据库混合云管理系统，可以同时管理云上和云下环境，在双 11 前快速把混合云构建起来，支撑双十一。第三，我们通过 VPC 网络连接阿里内部和公有云的网络，解决了混合云场景下的网络互联问题。

数据库弹性调度

使用云的资源还不够，为了实现更加极致的弹性能力，我们通过离在线混部技术，可以让数据库使用离线集群的计算资源，最大程度的降低成本。为了实现离在线混部技术，有两大基础条件：第一是容器化，通过容器实现了计算节点的资源隔离和统一调度，第二是计算存储分离，它是数据库弹性调度能力的基础。非常幸运的是，这几年技术的发展让存储计算分离成为可能，比如：25G 高速网络、RDMA 技术，高性能分布式存储等。

数据库存储计算分离架构如图，包括存储层、网络层和计算层，存储使用阿里自研分布式存储系统 – 盘古，数据库计算节点则部署在阿里自研容器（Pouch）中，通过 25G 网络与存储节点连接。

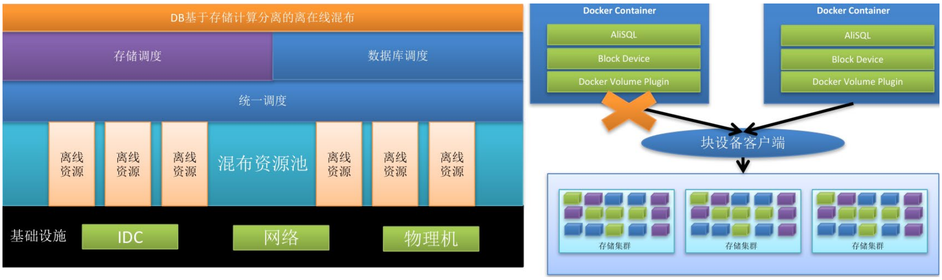
为了实现数据库存储和计算分离，我们在分布式存储 – 盘古上做了非常多的优化，比如：

- 响应延时：单路读写响应延时 0.4ms，RDMA 网络响应延时小于 0.2ms；
- 二三异步：第三个数据副本异步完成，极大提升了延时的稳定性；
- QoS 流控：根据前台业务负载情况控制后台 IO 流量，保证写入性能；
- 快速 Failover：存储集群单机 failover 优化为 5 秒，达到业界领先水平；
- 高可用部署：单集群四 Rack 部署，将数据可靠性提升到 10 个 9。

同时，在数据库方面我们也做了大量优化，最重要的是降低计算节点和存储节点的网络传输量，以此来降低网络延迟对于数据库性能的影响。第一是 redo log sync

优化，将数据库吞吐提升了 100%。第二是由于盘古支持原子写功能，所以我们关闭了数据库的 Double Write Buffer，高压力下数据库吞吐提升 20%，网络带宽节省了 100%。

双 11 数据库混部技术



容器化和存储计算分离，使得数据库无状态化，具备调度能力。在双 11 高峰，通过将共享存储挂载到不同的计算集群（离线集群），实现数据库的快速弹性。

阿里新一代数据库技术

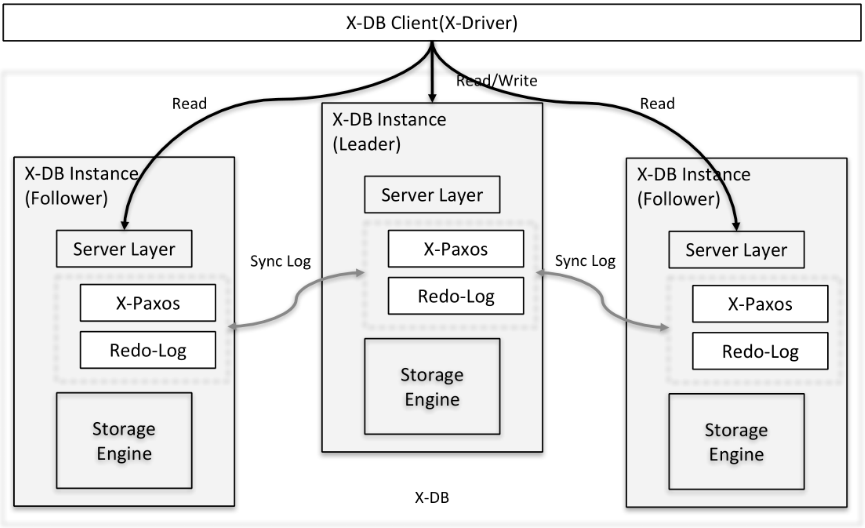
阿里最早是商业数据库，然后我们做去 IOE，研发出阿里 MySQL 分支 AliSQL 和分布式中间件 TDDL。2016 年，我们开始研发阿里新一代数据库技术，我们把它命名为 X-DB，X 代表追求极限性能，挑战无限可能的含义。

阿里的业务场景对于数据库有很高的要求：

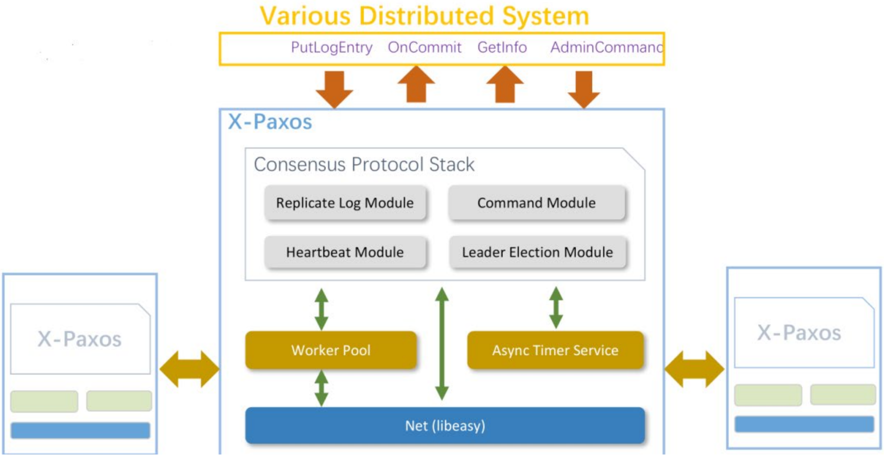
- 数据要可扩展；
- 持续可用、数据要强一致；
- 数据量大、重要程度高；
- 数据有明显的生命周期特性，冷热数据特点鲜明；
- 交易、库存，支付等业务，操作逻辑简单，要求高性能。

因此，定义新一代数据库就要包含几个重要特点：具备数据强一致、全球部署能力；内置分布式、高性能、高可用能力；具备自动数据生命周期管理能力。

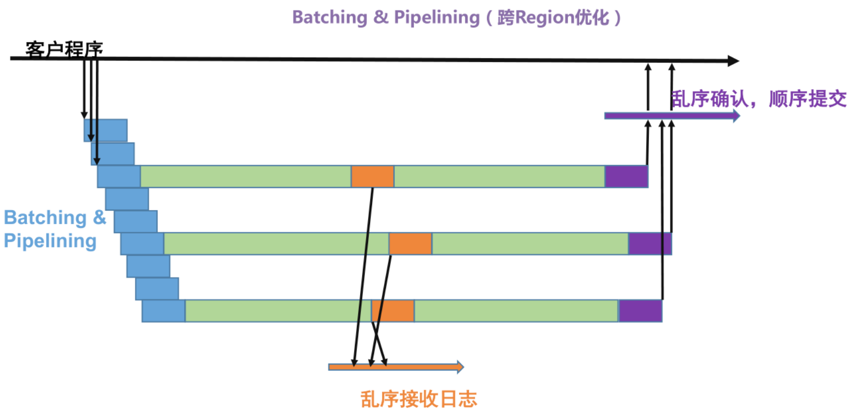
X-DB 架构图



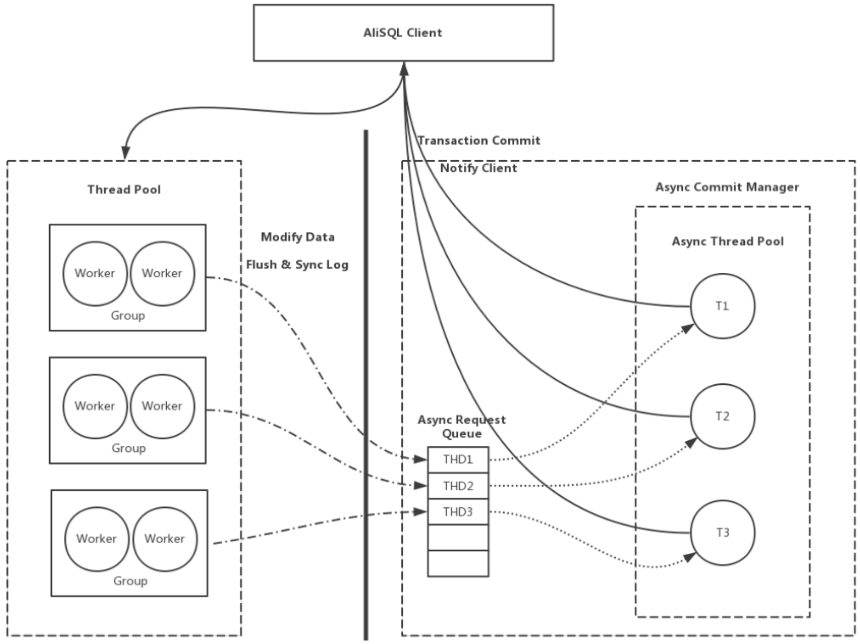
X-DB 架构如图，引入 Paxos 分布式一致性协议解决问题；可异地部署，虽然网络延时增加，但能够保持高性能（吞吐），在同城三节点部署模式下，性能与单机持平，同时具备网络抖动的高容忍性。



X-DB 核心技术之一：高性能 Paxos 基础库 X-Paxos 是实现三节点能力的核心，可实现跨 AZ、Region 的数据强一致能力，实现 5 个 9 以上的持续可用率。



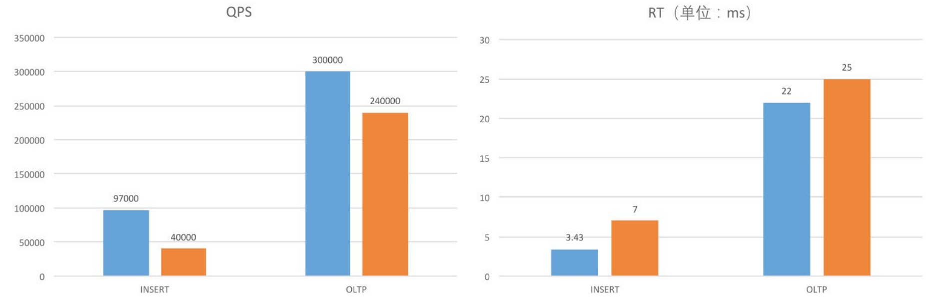
X-DB 核心技术之二：Batching & Pipelining。X-DB 在事务提交时，必须保证日志在数据库节点的多数派收到并提交，这是保证数据强一致基础，由于事务在提交时必须需要跨网络，这一定会导致延时增加，要保证高延时下的吞吐是非常困难的。Batching & Pipelining 技术保证尽可能批量提交，数据可以乱序接收和确认，最终保证日志顺序提交。可以在高延时的情况下，保持很高的吞吐能力。



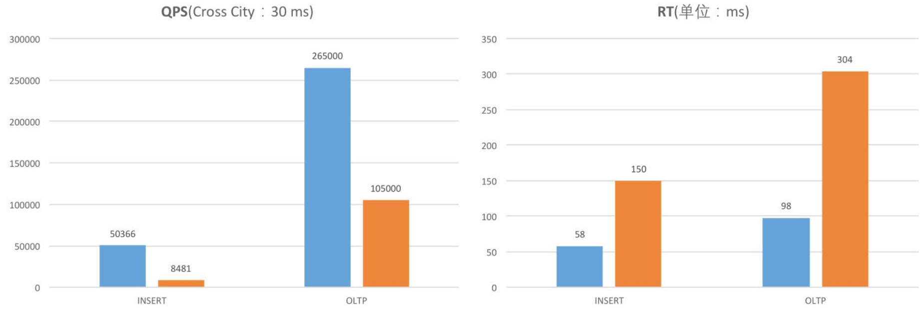
X-DB 核心技术之三：异步化提交，数据库线程池在提交时会等待，为了最大程度提升性能，我们采用了异步化提交技术，最大可能保证数据库线程池可以高效工作。通过这些技术保证 X-DB 在三节点模式下的高吞吐量。

X-DB 与 MySQL Group Replication 对比测试

X-DB vs MySQL GR

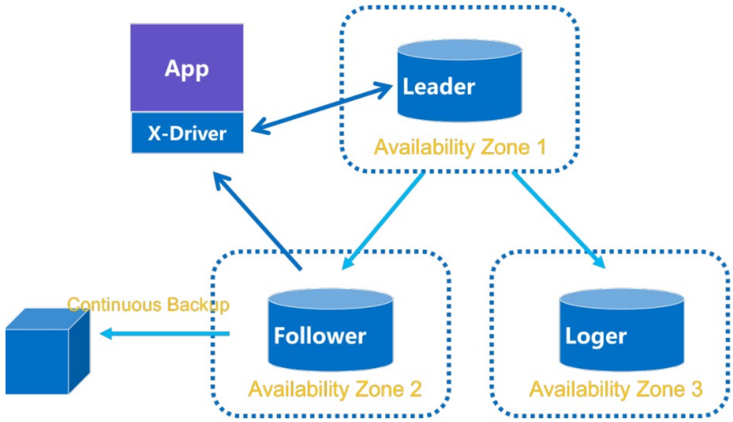


我们与 Oracle 官方的 Group Replication 作对比。在三节点同 IDC 部署模式下，sysbench 标准化测试。Insert 场景，我们可以做到 MySQL 官方的 2.4 倍，响应时间比官方低。

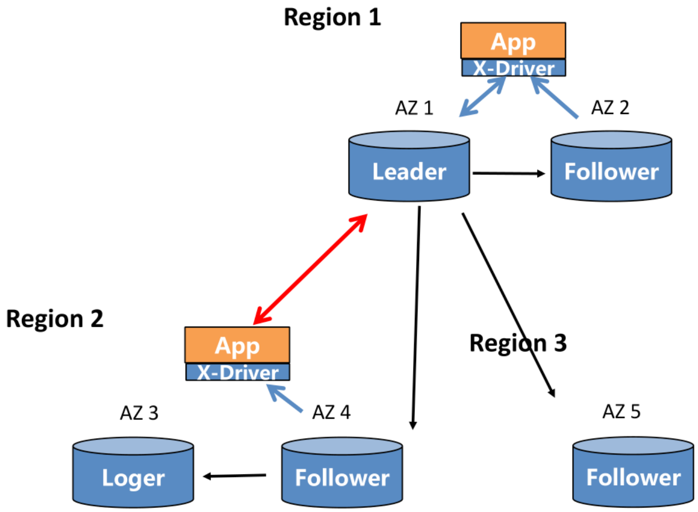


在异地部署模式下，sysbench 标准化测试。Insert 场景，X-DB（5.04 万）性能优势特别明显，是 MySQL GR（0.85 万）的 5.94 倍，响应延时 X-DB（58ms）是 MySQL GR（150ms）的 38%。

典型应用场景



同城跨 AZ 部署替代传统主备模式，我们把原来主备模式变成三节点，解决跨 AZ 数据质量问题和高可用问题。跨 AZ 数据强一致，单 AZ 不可用数据零丢失、单 AZ 不可用秒级切换、切换自封闭，无第三方组件。相对主备模式零成本增加。



跨 Region 部署，用更底层的数据库技术解决异地多活问题，三地六副本（主备模式）降低为三地五副本（三地五节点四数据），对于业务来说，可以享受跨 Region

数据强一致，单个 Region 不可用零数据丢失；跨 Region 强同步下依然保持高性能；切换策略灵活，可以优先切换同 Region，也可定制跨 Region 切换顺序。

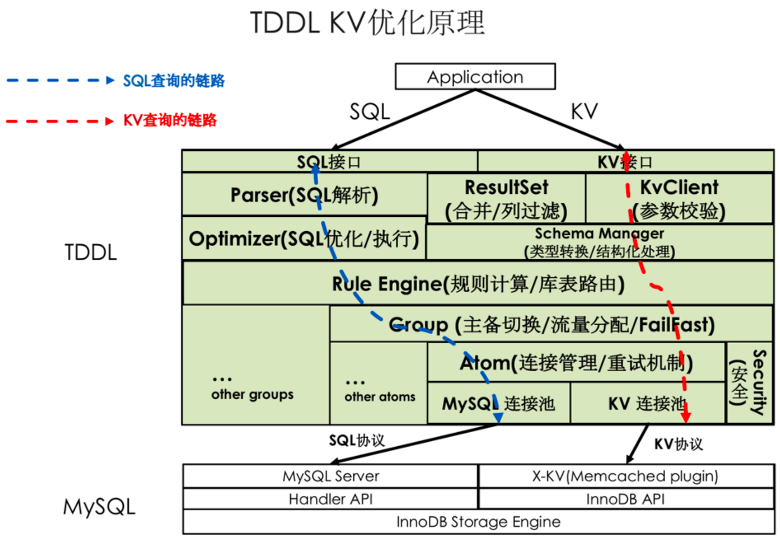
数据库在双 11 中的黑科技

X-KV 在双 11 中的应用

X-KV 是基于官方 MySQL Memcached plugin 的增强，今年我们做了大幅度的改进，支持更多数据类型，支持非唯一索引、组合索引，multi get 功能，还支持 Online Schema change。最大变化是通过 TDDL 支持 SQL 转换。对于业务方，X-KV 优势是超高读取性能，数据强一致，减少应用响应时间，降低了成本，同时因为支持 SQL，应用可以透明迁移，使用成本大幅降低。

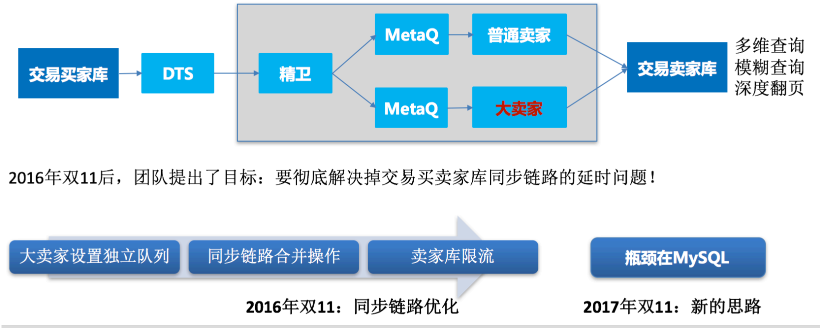
TDDLfor X-KV 实现了如下功能：

- 独立的连接池：SQL 和 KV 连接池相互独立；变更时，两套连接池保持协同一致；应用可以快速在两套接口之间切换。
- 优化的 KV 通信协议：不再需要分隔符，协议实现。
- 结果集自动类型转换：字符串自动转换为 MySQL 类型。



交易卖家库的性能瓶颈解决方案

随着双 11 交易量增长，近两年交易买家库和卖家库的同步延时一直比较大，导致商户不能及时处理双 11 订单；且卖家库有大量复杂的查询，性能差。我们曾经通过为大卖家设置独立队列、同步链路合并操作和卖家库限流等进行优化，但仍然没有完全解决问题。



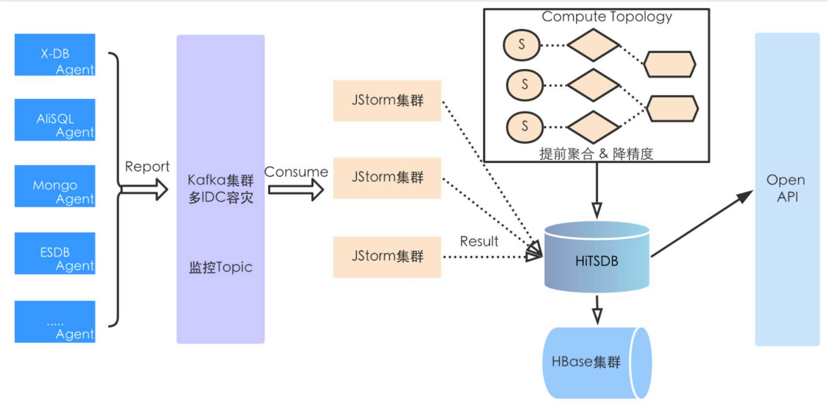
ESDB 是基于 ES 打造的分布式文档数据库，我们在 ElasticSearch 的基础上，支持了 SQL 接口，应用可以从 MySQL 无缝迁移到 ESDB；针对大卖家，提供动态二级散列功能，彻底解决了数据同步的性能瓶颈，而且 ESDB 还可以提供复杂的查询能力。

数据库监控系统演进

数据库监控系统的技术挑战具体有以下四点：

1. 海量数据：平均每秒 1000 万项监控指标，峰值 1400 万；
2. 复杂的聚合逻辑：地域、机房、单元、业务集群、数据库主备等多维度数据聚合；
3. 实时性要求高：监控盯屏需要立即看到上一秒的监控数值；
4. 计算资源：占用尽可能少的资源进行采集和计算。

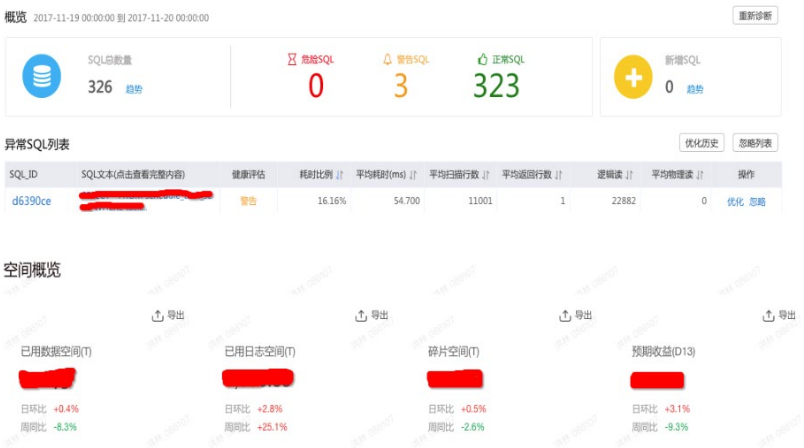
整个链路经历三代架构：第一代 Agent + MySQL；第二代 Agent + datahub + 分布式 NoSQL；第三代 Agent + 实时计算引擎 + HiTSDB



HiTSDB 是阿里自研的时序型数据库，非常适合存储海量的监控类数据。通过实时计算引擎将秒级性能数据、全量 SQL 运行状况进行预先处理后，存储在 HiTSDB 中。通过第三代架构，实现了双 11 高峰不降低的秒级监控能力，这对我们了解系统运行状况、诊断问题是非常有帮助的。

CloudDBA 在双 11 中的应用

阿里拥有业界最富有经验的 DBA，海量的性能诊断数据。我们的目标是把阿里 DBA 的经验、大数据和机器智能技术结合起来，目标是三年后不再需要 DBA 做数据库诊断、优化等工作，而是让机器来完成数据库的智能化管理。我们认为自诊断、自优化、自运维是未来数据库技术发展的重要方向。



CloudDBA 在今年双 11 也做了一些探索，通过对全量 SQL 以及监控数据的分析，我们实现了 SQL 自动优化（慢 SQL 调优）、空间优化（无用表无用索引分析）、访问模型优化（SQL 和 KV）和存储空间增长预测等功能。

展望明年双 11

展望明年的双 11，我总结了三个关键词：**Higher，Faster，Smarter**

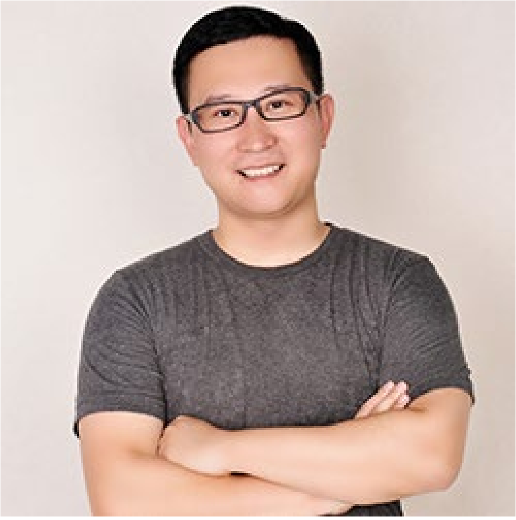
Higher 意味着更高的交易峰值，背后其实是更低成本的追求，用极致的弹性能力支持更高的峰值，给用户最好的购物体验，希望有一天可以做到不限流。

Faster 是我们技术人一直不变的追求，更快的应用系统、更快的数据库，更快的存储，更快的硬件等等。天下武功，唯快不破。

Smarter 是机器智能在双 11 中的应用，不管是数据库、调度、个性化推荐甚至客服等方面，我们都希望机器智能可以得到更多的应用，产生更大的技术突破。

争分夺秒：阿里实时大数据技术全力助战双 11

大沙



大沙，阿里巴巴高级技术专家，负责实时计算 Flink SQL，
之前在美国脸书任职，Apache Flink committer。

实时计算 in 阿里巴巴

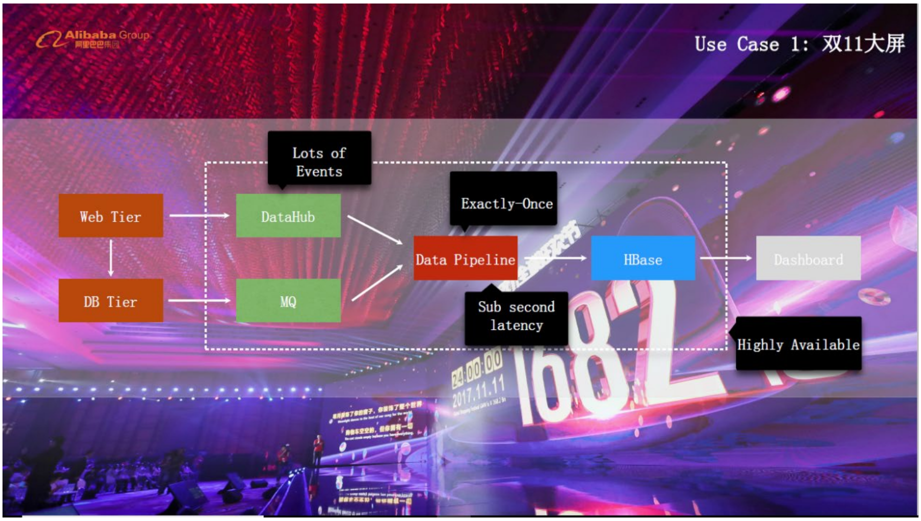
1999 年起，阿里从电商平台开始不断拓展业务，在金融、支付、物流、文娱各个领域衍生出众多产品，例如依托于淘宝、天猫为主的电商平台、阿里妈妈广告平台、蚂蚁金服支付宝、阿里云、大文娱等。今天的阿里它已经不仅仅是一个电商平台，而是一个庞大的应用生态。阿里巴巴目前是全球最大的电商平台，2016 财年收入达到 5500 亿美金。在阿里平台上有 5 亿的用户，相当于中国人口的 1/3，每天有近 1000 万用户通过阿里平台交易。

阿里俨然成为巨大的商业航母，在这艘航母之上，大量的用户和应用必然会产生大量的数据。目前，阿里巴巴的数据量级已经达到 EB 级别，每天的增长量达到 PB 级别，实时计算日常峰值处理的数据量可达到 1 亿每秒，今年双 11 更是达到了惊人

的 4.7 亿每秒。

实时计算在阿里巴巴内部应用广泛。随着新经济体的发展，技术的革新和用户需求的提升，人们越来越需要实时计算的能力，它的最大好处就是能够基于实时变化数据更新大数据处理的状态和结果。接下来，举两个例子来阐释实时计算在阿里内部应用的场景：

1. 双 11 大屏



每年双 11 阿里都会聚合有价值的数据展现给媒体，GMV 大屏是其中之一。整个 GMV 大屏是非常典型的实时计算场景，每条交易数据经过聚合展现在大屏之上。从 DataBase 写入一条数据开始，到数据实时处理写入 HBase，最后展现在大屏之上，整个过程的链路十分长。整个应用也存在着诸多挑战：

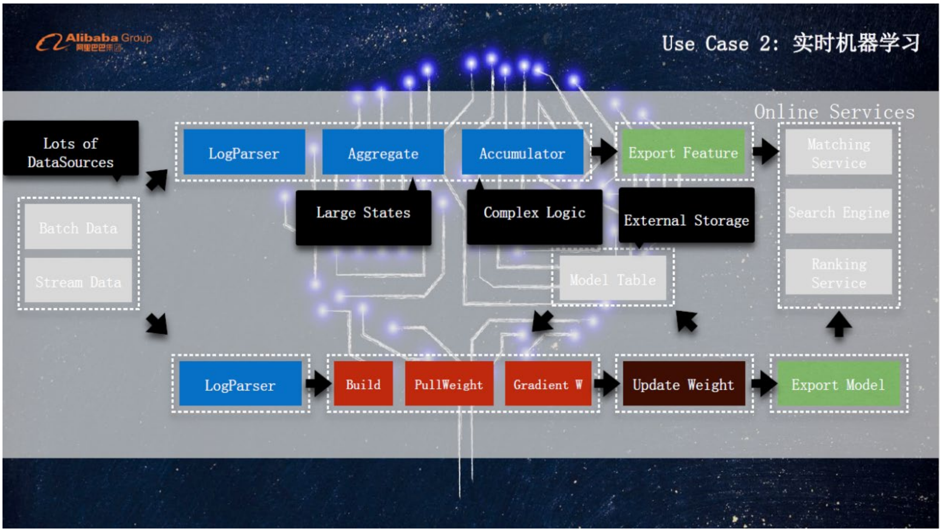
- 大屏展现需要秒级延迟，这需要实时计算延迟在亚秒级别
- 数据库产生的大量数据需要在一个 Job 中聚合完成
- 需要实现 Exactly-Once 以保障数据计算的精确性
- 系统高可用，不能有卡顿和不可用的情况

这个应用场景的 SLA 非常高，要求秒级延迟和数据的精确性，但它的计算并不

复杂，接下来介绍一个更为复杂的应用。

2. 实时机器学习

机器学习一般有两个重要的组件：Feature 和 Model。传统的机器学习使用批计算对 Feature 的采集和 Model 的训练，这样更新频率太低，无法适应数据在不断变化的应用的需求。例如在双 11 时，商品的价格、活动的规则与平时完全不同，依据之前的数据进行训练得不到最优的效果。因此，只有实时收集 Feature 并训练 Model，才能拟合出较为满意的结果。为此，我们开发了实时机器学习平台。

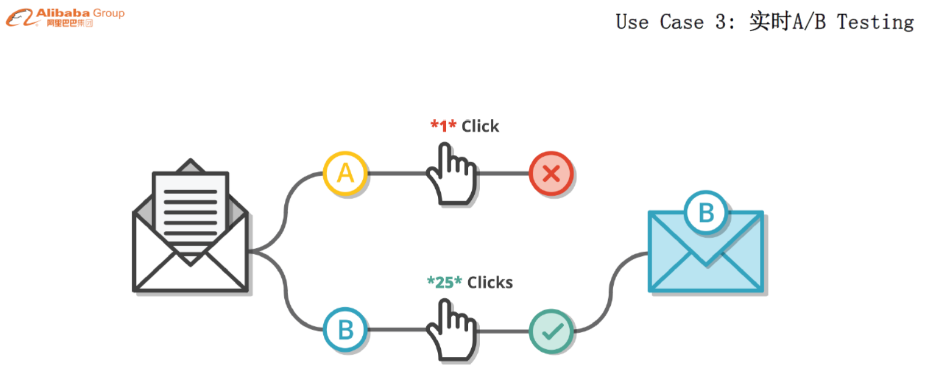


实时机器学习平台主要包括两个部分：实时 Feature 计算和实时 Model 计算。这套系统同样拥有很多挑战，具体如下：

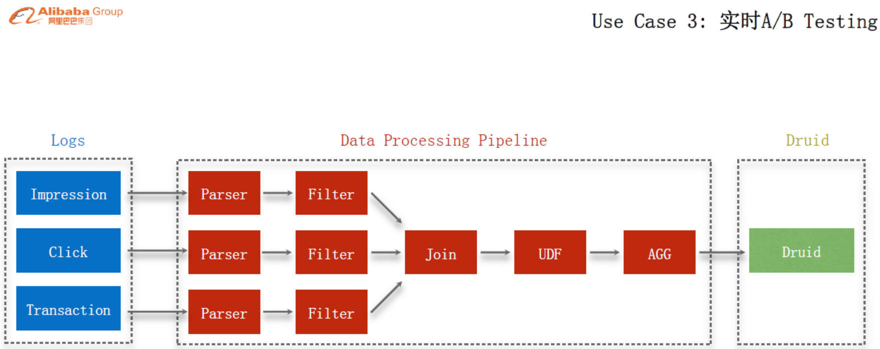
- 机器学习需要从诸多数据源采集各式各样的 Metrics
- 维度多。维度的叠加有可能是用户维度和商品维度等等的笛卡儿积的，导致最后的 Metrics 是海量的，实时计算的存储 State 将是异常巨大的
- 机器学习计算复杂，会耗用大量的 CPU 计算资源
- 某些数据不能存在 State 中，需要外部存储，在计算过程中会引入大量的外部 IO 读取

3. 实时 A/B Testing

用户的 Query 也有可能是基于实时数据的不断变化的，典型的例子有实时的 A/B Testing。



算法工程师在调优 Model 时会测试并对比多种 Model，不同的 Model 有不同的计算模式和方法，产生不同的计算结果。因此，往往会有不同的 Query 订阅实时数据，产生结果后根据用户回馈迭代 Model，最终得到最优模型。A/B Testing 的挑战在于算法工程师往往需要计算很多 Metrics。如果所有的 Metrics 都通过实时计算进行统计，会浪费大量资源。



针对这个挑战，我们设计了 A/B Testing 的框架开发平台。它用来同步算法工程师感兴趣的 Metrics 进行聚合，收集起来并发送到 Druid 引擎。这样，算法工程师根

据不同 Job 的要求清洗数据到 Druid，最后基于 Druid 对不同的 Metrics 进行统计分析，从而找到最有的算法 Model。

综上，实时计算在阿里巴巴内部存在如下挑战：

- 业务庞大，场景多，大量的机器学习需求，这些因素一起导致了计算逻辑十分复杂
- 数据量大，作业多，从而整个实时计算的机器规模十分巨大
- 要保障低延迟和数据精确性，同时要满足高吞吐量的需求

Flink 的选定及阿里 Blink

为了应对上述挑战，我们调研了许多计算框架，最终选定 Flink，原因如下：

1. Flink 很好地引入和设计了 State，基于 State 复杂的逻辑计算如 join 和 aggregate 能被很方便的实现
2. Flink 引入了 Chandy-Lamport 算法，在此算法的支撑下可以完美实现 Exactly-Once 等语义，并能在低延迟下实现高吞吐计算。

然而，Flink 在 State、Chandy-Lamport 算法等这些方面还有一些缺陷，为此阿里开辟了名为 Blink 的项目。

Blink是开源Apache Flink与阿里巴巴Improvement的结合，主要分两大块：

1. Blink Runtime

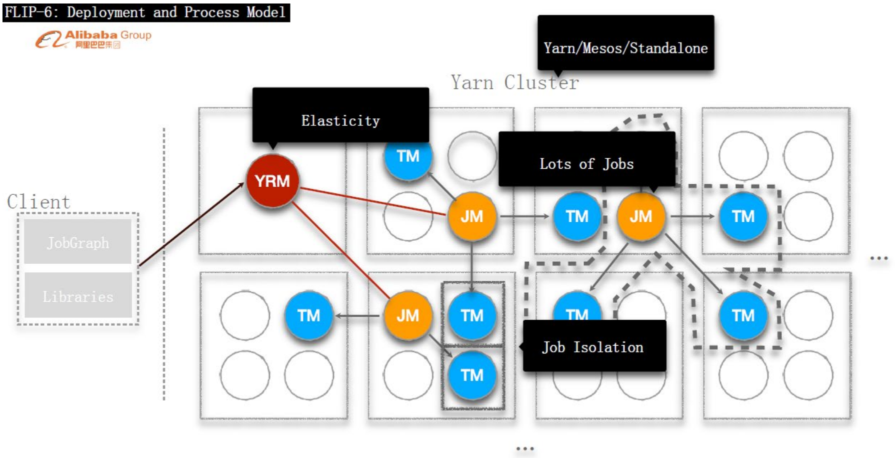
主要包括存储、调度和计算，不同公司在使用 Flink 时，存储、调度以及底层优化等方面会有诸多不同，阿里巴巴的 blink 内部也对 Runtime 做了诸多个性化的优化，这一层不好与 Apache Flink 社区统一，我们称之为 Blink Runtime。

2. Flink SQL

原生的 Flink 只有比较底层的 DataStream API，用户在使用时需要设计实现大量的代码，次外 DataStream 本身也有设计上的缺陷。为了方便用户使用，阿里巴巴团队设计了流计算的 Flink SQL 并推回了社区。取名 Flink SQL 而不是 Blink SQL，主要原因 Blink 和 Flink 在 SQL 这个用户 API 上面是完全和社区统一的，另外 Apache Flink 的大部分功能都是阿里巴巴贡献的，所以说 Flink SQL 就是 Blink SQL，没有特别大的区别。

Blink Runtime 核心优化解密

1. 调度和资源管理的优化



这部分的优化主要包含以下几点：

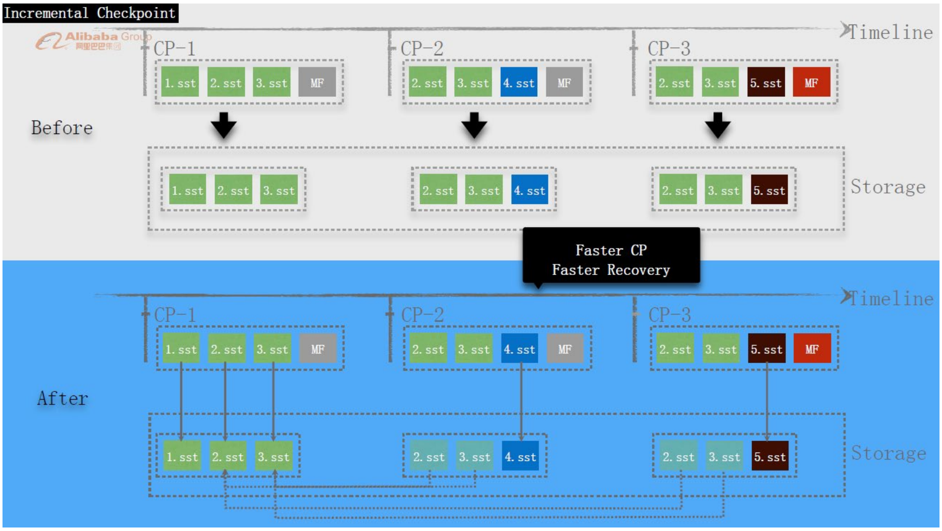
- Blink Runtime 解决了大规模部署问题。早期的 Flink 中一个 Cluster 只有一个 JobMaster 来管理所有的 Job。随着 Job 的不断增加，单一的 Master 无

法承接更多的 Job，产生了瓶颈。因此，我们重构了架构，使每一个 Job 拥有自己的 Master。

- 早期的 Flink 中 TaskManager 管理很多 Task，某一个 Task 的问题有可能导致整个 TaskManager crash，进而影响其他 Job。我们改变了设计，使得每一个 Job 都拥有自己的 TaskManager，实现了 Job 的隔离。
- 引入 ResourceManager。ResourceManager 可以和 JobMaster 通讯，实时动态地调整资源，达到更优的集群资源分配。
- 我们不仅将这些优化应用在 YarnCluster 上，还应用到 Mesos 和 Standalone 等部署模式上。

有了这些工作，Flink 就可以应用到大规模的集群部署中，支撑成千上万的 job。

2. Incremental Checkpoint

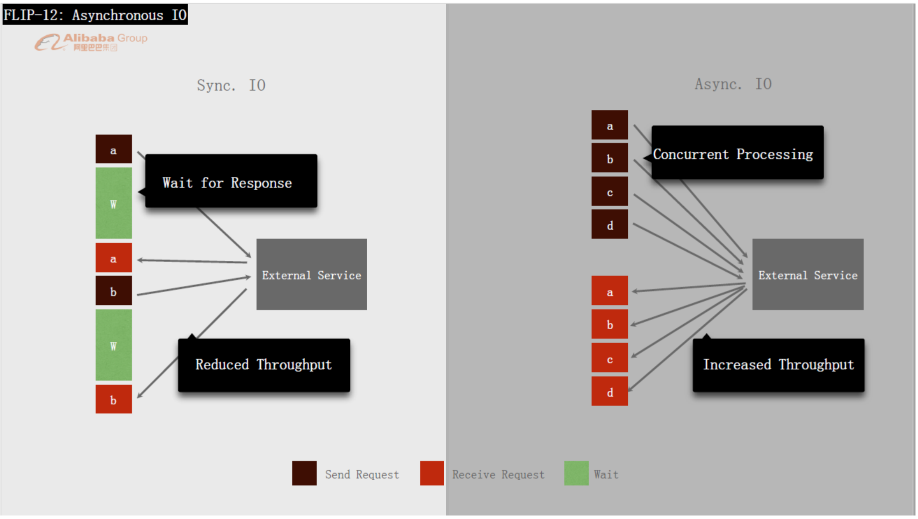


实时计算需要不停的在 checkpoint 的时候来保留计算状态。早期的 Flink 的 checkpoint 的设计存在缺陷，在每个 checkpoint 发生的时候，它会读取所有旧的状态数据，和新的数据合并后按照全量的方式写入磁盘。随着 State 的不断增大，在每次做 checkpoint 的时候所需要的数据读取和写入的量级是十分巨大。这就导致 Job

的 checkpoint 的间隔需要设置的很大，不能小于 1 分钟。越大的 checkpoint 的间隔，failover 的时候回退的计算就越大，造成的数据延迟也就越严重。

为了减少 checkpoint 间隔，我们提出了 Incremental Checkpoint 的设计。概括的说就是在 checkpoint 的时候只存储增量的 state 变化的数据。由于历史上每个 checkpoint 的数据都已经保存，后面的 checkpoint 只需要将不同的数据放入存储，这样每次做 checkpoint 需要更新的数据量就非常小，使得 checkpoint 可以在若干秒级内完成，这就大大减小了 failover 时可能引起的延迟。

3. 异步 IO



很多时候我们不得不将数据放在外部存储中，这样在计算过程中就需要通过网络 IO 读取数据。传统的方式使用 Sync-IO 的读取方式，在发出数据请求之后，只有等待到结果返回之后才能开始下一个数据请求，这种做法造成了 CPU 资源的浪费，因为 CPU 在大多数情况下都在等待网络 IO 的请求返回。Sync-IO 使得 CPU 的资源利用率无法提高到极致，也就大大影响了单位 CPU 下的计算吞吐。为此提升计算吞吐，我们设计了 Async-IO 的数据读取框架，它允许异步地多线程地读取数据。每次数据请求发出后不需要等待数据返回就继续发送下一个数据请求。当数据请求从外部存储返回后，计算系统会调用 callback 方法处理数据。如果数据计算不需要保

序，数据返回之后就会快速经过计算发出。如果用户需要数据的计算保序时，我们使用 buffer 暂时保存先到的数据，等前部数据全部到达后再批量地发送。在使用了 Async-IO 之后，根据设置的 buffer 大小不同计算吞吐可以提升几十倍甚至几百倍，这就极大地提升了单位 CPU 利用率和整体的计算性能。

值得一提的是，以上所述的所有 Blink Runtime 优化已经全部贡献给了 Apache Flink 社区。

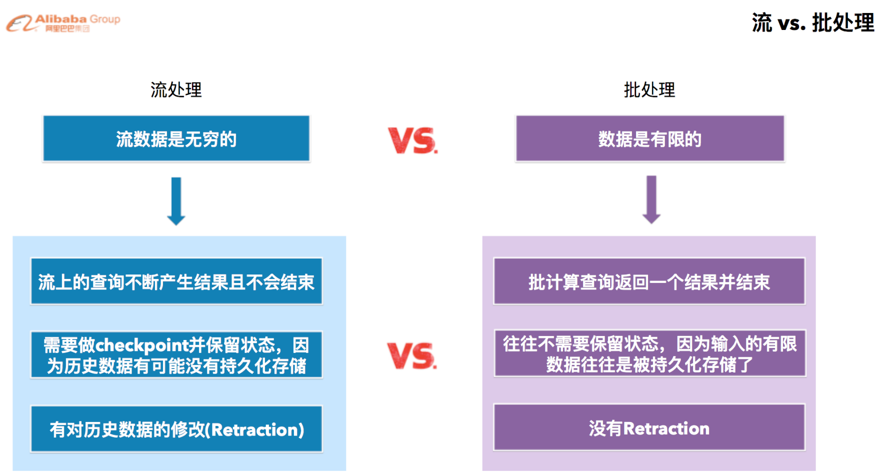
Flink SQL 核心功能解密

1. 阿里贡献了 Apache Flink SQL 80% 的研发工作

目前，Apache Flink SQL 80% 的功能是阿里巴巴实时计算团队贡献的，包括两百个提交和近十万行代码。使用 Flink SQL 的原因是因为我们发现了底层 API 给用户的迁移、上线带来的极大不便。那么，我们又为什么选择 SQL ？主要原因如下：

- SQL 是十分通用的描述性语言，SQL 适合用来让用户十分方便的描述 Job 的需求。
- SQL 拥有比较好的优化框架，使得用户只需要专注于业务逻辑得设计而不用关心状态管理，性能优化等等复杂得设计，这样就大大降低了使用门槛。
- SQL 易懂，适合不同领域的人使用。使用 SQL 的用户往往都不需要特别多的计算机编程基础，从产品设计到产品开发各种人员都可以快速掌握 SQL 的使用方法。
- SQL 的 API 十分稳定，在做机构升级，甚至更换计算引擎时都不用修改用户的 Job 而继续使用。
- 有些应用场景需要流式更新，批式验证。使用 SQL 可以统一批计算和流计算的查询 query。真正实现一个 Query，同样的结果。

2. 流处理 VS 批处理



要想设计和批处理统一的流计算 SQL，就要了解流处理和批处理的区别。两者的核心区别在于流处理的数据是无穷的而批处理的数据是有限的。这个本质区别又能引入另外三个更具体的区别：

- 流处理会不断产生结果而不会结束，批处理往往只返回一个最终结果并且结束。比方说，如果要统计双 11 的交易金额，使用批处理计算就要在双 11 当天的所有交易结束后，再开始计算所有买家花费的总金额并得到一个最终数值。而流处理需要追踪实时的交易金额，实时的计算并更新结果。
- 流计算需要做 checkpoint 并保留状态，这样在 failover 的时候能够快速续跑。而批计算由于它的输入数据往往是被持久化存储过的，因此往往不需要保留状态。
- 流数据会不断更新，例如某一买家的花费总金额在不断变化，而批处理的数据是一天花费的总金额，是固定的，不会变化的。流数据处理是对最终结果的一个提前观测，往往需要把提前计算的结果撤回（Retraction）做更改而批计算则不会。

3. Query Configuration

上面提到的这些区别都不涉及用户的业务逻辑，也就是说这些区别不会反应在 SQL 的不同。我们认为这些区别只是一个 job 的属性不同。为了描述流计算所特有的一些属性，例如什么时候产生流计算结果和怎么保留状态，我们设计容许用户配置的 Query Configuration，它主要包括两个部分：

1. Latency SLA

   定义了从数据产生到展现的延迟，如双 11 大屏是秒级别。用户根据自己的需要配置不同 SLA，我们的 SQL 系统会根据 SLA 的要求做最好的优化，使得在满足用户需求的同时达到系统性能的最优。

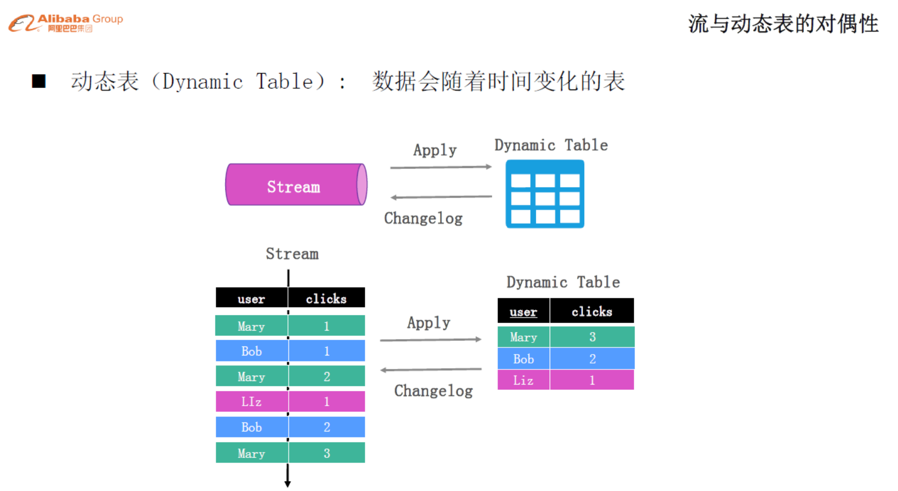
2. State Retention/TTL

   流计算是永不停止的，但是流数据中的 State 往往不需要保留很久，保留过久势必对存储是个浪费，也极大的影响了性能。所以我们容许用户设置合理的 TTL（过期时间）来获得更好的计算性能。

我们通过 Query Configuration 描述了流和批所不同的一些属性。接下来我们需要继续考虑如何设计流式的 SQL？

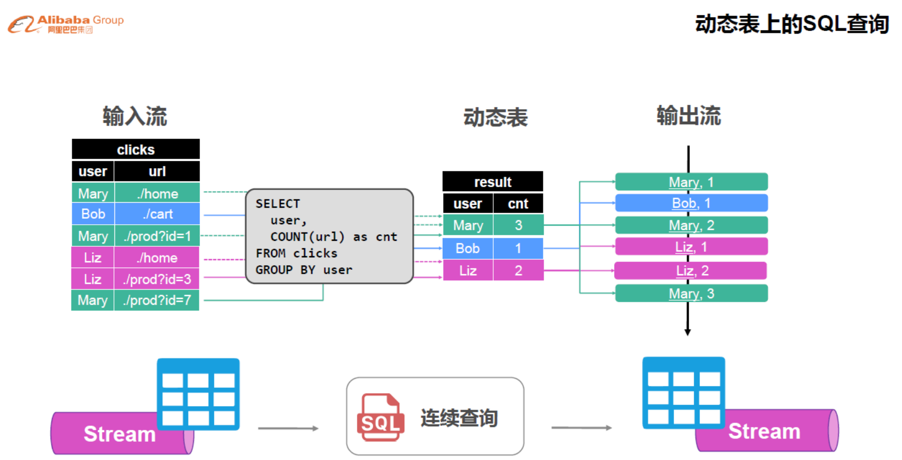
4. 动态表（Dynamic-Table）

众所周知，传统的 SQL 是基于表的操作，而流数据中并没有表，这就使得设计流计算 SQL 举步维艰。因此，我们提出了数据会随着时间而变化的动态表 (Dynamic-Table) 的概念 (http://flink.apache.org/news/2017/04/04/dynamic-tables.html)。动态表是流的另一种表现形式，它们之间具有对偶性，即它们可以互相转换而不破坏数据的一致性。以下是一个例子：



如图，左边是输入流，我们为每一条数据产生 Dynamic-Table，再将 Table 的变化用 Changelog 发送出去。这样两次变化后，输入流和输出流中的数据始终保持一致，这就证明了引入 Dynamic-Table 并没有丢失语义和数据。

有了动态表的概念，我们就可以应用传统 SQL 作用于流上。值得一提的是，Dynamic-Table 是虚拟的存在着，它并不需要实际的存储来落地。我们再来看一个例子：

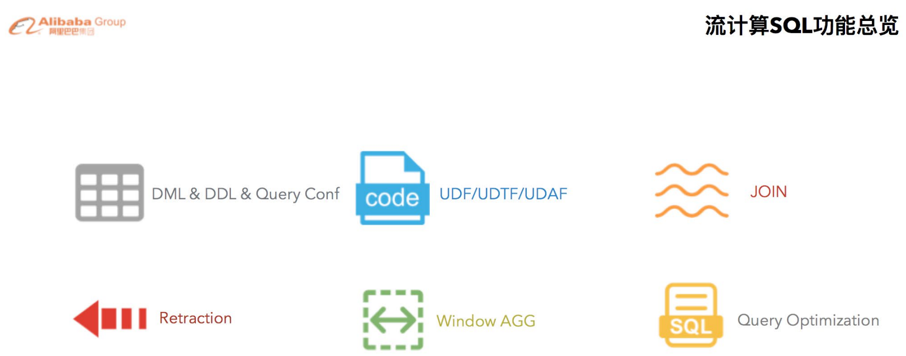


如图，当有输入流的时候我们进行连续查询。我们将 Stream 理解为一个 Dynamic-Table，动态查询是基于 Dynamic-Table 产生一个新的 Dynamic-Table，如果需要新产生的 Dynamic-Table 还可以继续产生流。这里，因为加入了连续查询的聚合计算，左右两边的流已经发生了变换。总之动态表的引入提供了我们在流上做连续 SQL 查询的能力。

5. Stream SQL 是没必要存在的

通过上面的讨论，我们发现有了 Dynamic-Table 之后我们不需要再创造任何新的流式 SQL 的语义。因此我们得出这样的结论：流式 SQL 是没必要存在的。ANSI SQL 完全可以描述 Stream SQL 的语义，保持 ANSI SQL 的标准语义是我们构建 Flink SQL 的一个基本原则。

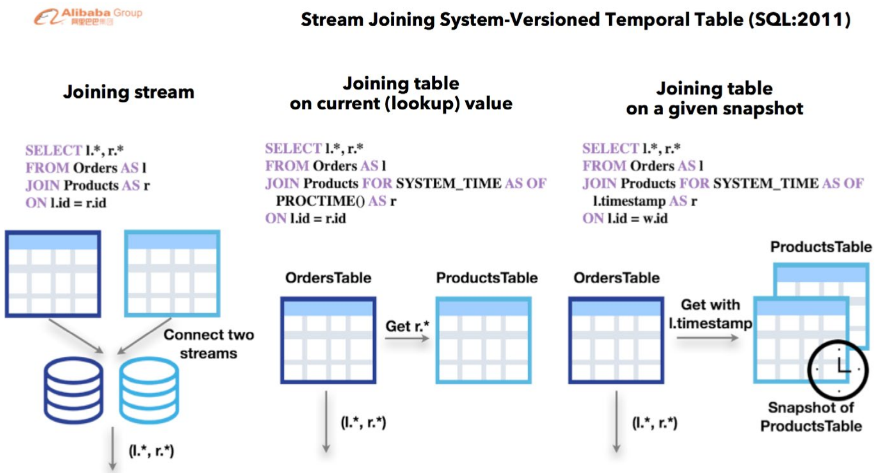
6. Flink SQL 功能简介



基于上面的理论基础，我们继而实现了流计算所需要的若干 ANSI SQL 功能，包括：DML、DDL、UDF/UDTF/UDAF、连接 Join、撤回 (Retraction)、Window 聚合等等，除了这些功能之外，我们还做了大量的查询优化，从而保障了 Flink SQL 即能满足用户的各种查询的需求，同时兼具优异的查询性能。接下来，简要介绍其中几项：

1) JOIN

流和动态表具有对偶性，一条 SQL 看似是 Table 的 join，事实上是流的 join。



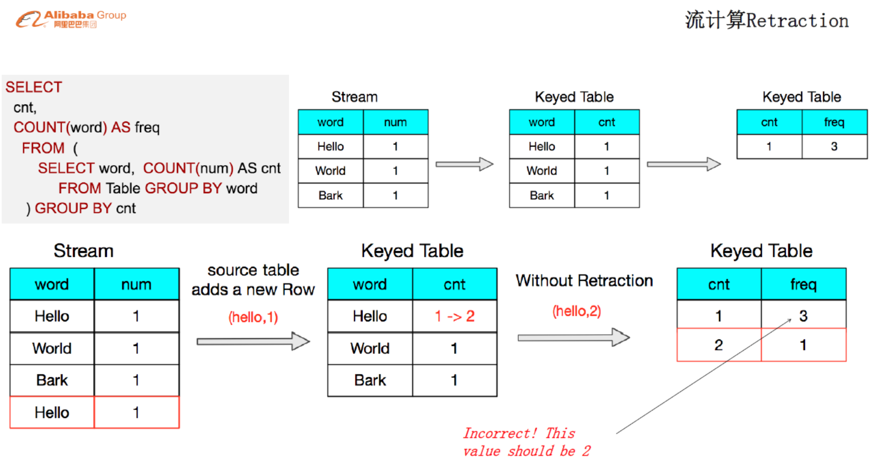
例如 Inner Join 的实现原理如下：数据会从输入的两边任意一条流而来，一边数据先来会被存在 State 中并按照 Joining key 查询另外一边的 State，如果存在就会输出结果，不存在则不输出，直到对面数据来了之后才产生结果。总之，两个流具有两个 state，一边的数据到达后存下来等待另外一边数据，全部到达后 inner join 产生结果。除了两条流的 join 之外，我们还引入了流和外部表的 join。我们的机器学习平台会把大量的数据存储在 HBase 中，查询 HBase 中的数据的操作实际上是在连接一个外部表。连接外部表往往存在两个模式：

- Look up 方式。流数据到达时即时地查询外部表，从而得到结果。
- Snapshot 方式。流数据到达时即时地发送 snapshot 的版本信息给外部存储 service 从而查询数据，外部表存储根据版本信息返回结果。

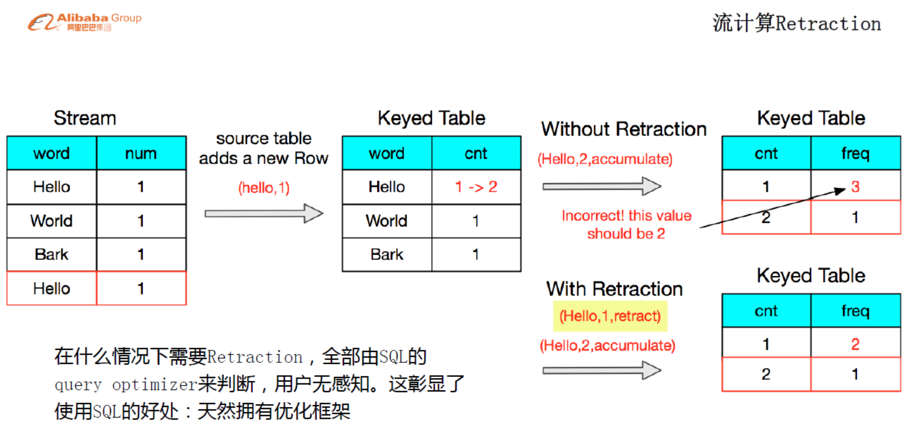
值得一提的是，我们设计的这个流和外部表关联的这个功能没有引入任何新的语法，是完全按照 SQL-2011 的标准实现的。同样的查询在批计算上也适用。

2) Retraction

撤回是流计算的一个重要概念，举一个例子作解释：计算词频

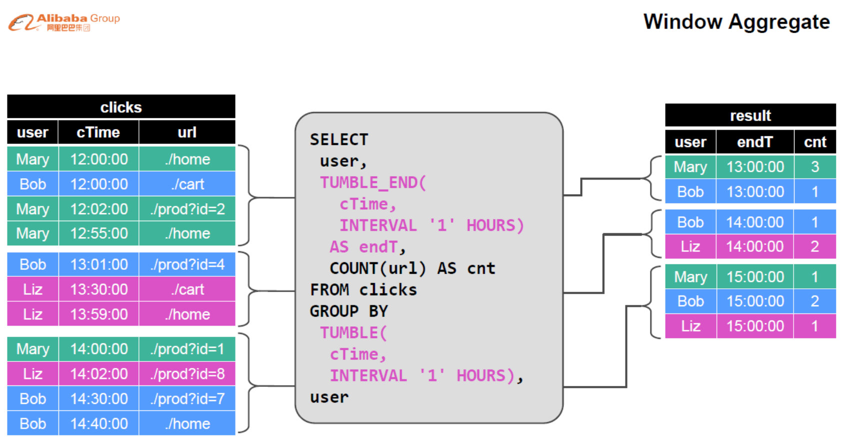


词频的计算是指对所有英文单词统计频率，并最终按照频率统计不同频率下不同单词的个数。例如，如果一个统计的初始状态只有 Hello World Bark 三个单词，且每个单词只出现一次，那么词频的最终结果就是出现频率为 1 的单词有 3 个（出现频率为其他次数的完全没有），因此结果表只有一行“1——3”。当单词不断更新，再增加一个 Hello 时，因为 Hello 的出现频率变为 2 次，我们在词频的结果表中插入“2——1”这么一行新的数据。显然，出现两次的单词是一个，那么“2——1”这个结果是对的，但是出现频率为 1 次的单词数已经错了，应该是 2 个，而不是 3 个。出现这种问题的本质原因是因为流计算输出的结果是对计算的一个提前观测，随着数据的不断更新，计算结果必然会发生改变，这就要求我们对之前发生的结果做撤回（retraction）再把更新的结果发出去，不然数据结果就不错误。对于上面的例子，当 Hello 的频率从 1 变到 2 的时候，我们不仅需要在结果表中插入“2——1”这么一行，还需要对“1——3”这一行做撤回更新操作。



值得一提的是什么时候需要撤回，什么时候不需要，完全由 SQL 的 Query Optimizer 来判断，这个用户是完全不需要感知的，用户只需要通过 SQL 描述他的业务计算逻辑就好了。如图所示，第一个场景不需要撤回而第二个需要，这完全是由优化框架决定而非用户。这一点，大大体现了使用 SQL，并利用 SQL 中所拥有的天然优化框架的好处。

3) Window 聚合

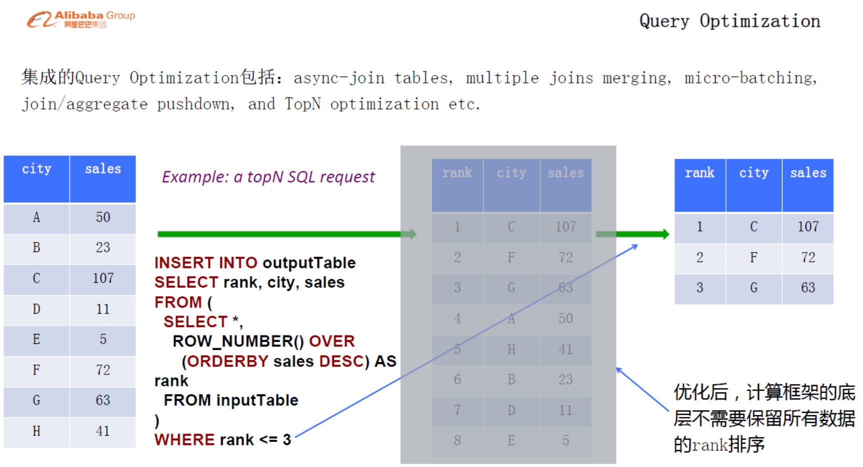


Window 聚合是 Flink SQL 的一个重要能力。图中的这个例子我们对每一个小时的数据做聚合统计。除了这种 Tumble window 我们还支持了 Sliding Window 和

Session Window。Window 的聚合事实上是按照用户给定的 Window 的边界做一个个小 batch 处理。

4) 查询优化 Query Optimization

除了添加新的功能，我们还做了大量的查询优化。例如 micro-batching。如果没有 micro-batching，处理每一条数据就会伴随着几次 IO 读写。有了 micro-batching 之后我们可以用几次 IO 处理来处理上千条数据。除此之外，我们还做了大量的的 filter/join/aggregate pushdown 以及 TopN 的优化，下面再举例解释 TopN 的优化：



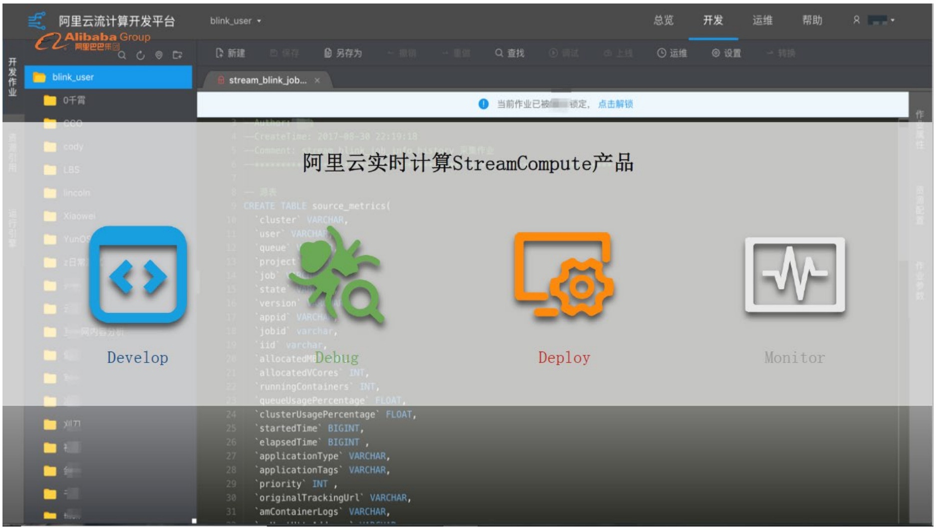
如上图，我们想取销售量前三的 city，对用户的 Query 有两种底层的实现：

- 一种方式是当没一条数据来的时候，对保存的所有 city 进行排序，再截取前三个 city。这种设计每条数据跟新都会重新排列所有 city，势必会造成大量计算资源浪费。
- 我们的 Query Optimizer 会自动识别到查询语句，对这种计算做优化，真正执行过程中只需要不停的更新排面前三的 city 就可以了，这样大大优化了计算的复杂度，提升了性能

阿里巴巴实时计算应用

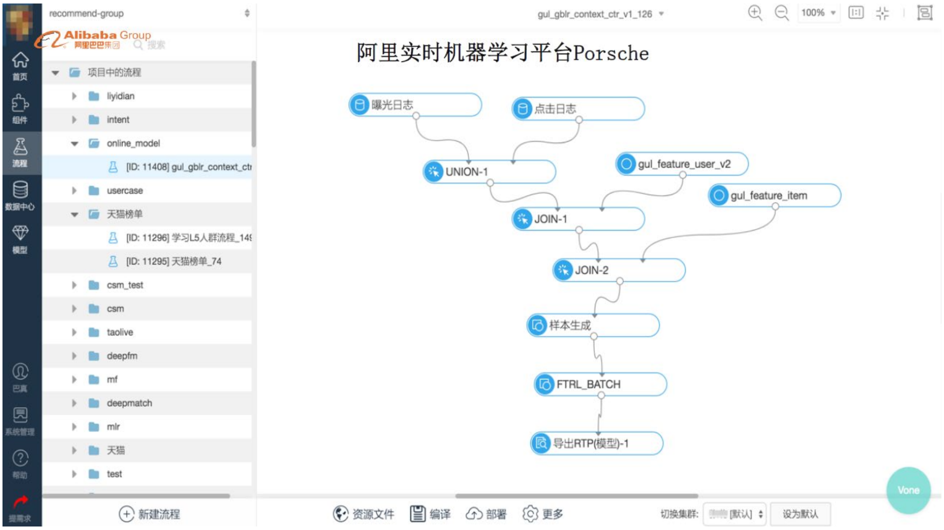
基于流计算 SQL 之上我们开发了两个计算平台。

1. 阿里云流计算平台



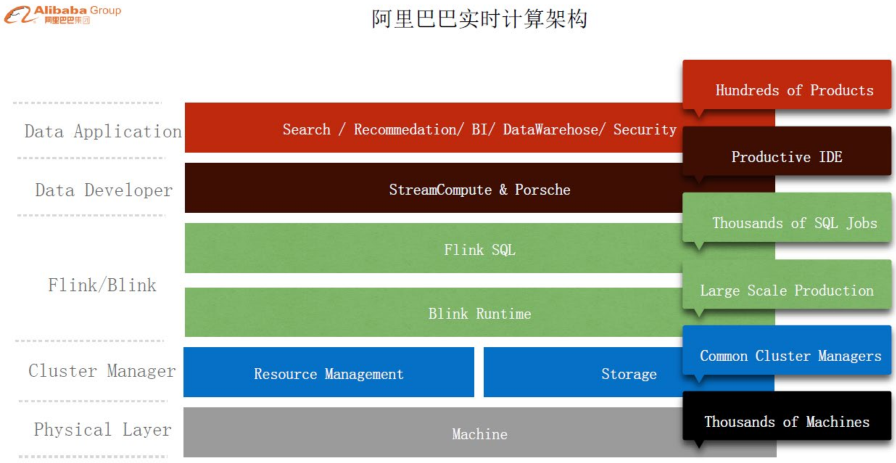
一个是阿里云流计算平台（streamCompute），该平台允许用户编写 SQL，并在平台内部调试 debug。调试正确后，用户可以通过这个平台直接将作业发布在阿里云集群上部署，部署完成后后检测运维上线的。因此这个平台整合了所有实时计算的需求，集开发、Debug、上线部署、运维于一体，大大加速了用户开发和上线的效率。值得一提的是，2017 年双 11 期间阿里集团绝大多数的实时计算 Job 均通过这个平台发布。我们今年 9 月开始，通过阿里云，包括公共云、专有云也将这个平台开放给外部企业，让他们能够使用到阿里巴巴实时计算的能力。

2. 阿里实时机器学习平台 Porsche



为了方便算法同学开发机器学习任务，我们基于 Flink SQL 以及 Hbase，设计实现了一个面向算法人员、支持可视化自助开发运维的在线机器学习平台——Porsche。如上图所示，用户在 Porsche 平台的 IDE，通过可视化的方式将组件拖入画布中，配置好组件属性，定义好完整的计算 DAG。这个 DAG 会被翻译成 SQL，最终提交给 Blink 执行。另外，值得一提的是，Porsche 平台还支持 Tensorflow，今年双 11 也是大放异彩，本平台免去了算法同学学习使用 SQL 的成本，暂时只对内开放。

双 11 实时计算总结



上图是阿里巴巴实时计算架构，底层是数千规模的物理机，之上是统一部署的 Resource Management 和 Storage，然后是 Blink Runtime 和 Flink SQL，用户通过 StreamCompute 和 Porsche 平台提交 Job，现在已经在阿里内部支持了数百个工程师近千个 Flink SQL Job。上述就是阿里巴巴实时计算的现状。

在实时计算的助力下，双 11 拿到 1682 亿的辉煌战果，实时计算的贡献主要体现在以下几点：

1. 本次双 11 是互联网历史最大规模的并发，每秒几十万的交易和支付的实时聚合统计操作全部是是由 Blink 计算带来的
2. 3 分 01 秒 100 亿数据的展现不仅需要高 Data Base 的高吞吐能力，还考验着实时计算的速度
3. 算法平台帮助算法同学取得了很好的搜索和推荐效果，获得了整体 GMV 的增长

总之，实时计算不仅满足了阿里巴巴内部多种多样的需求，还提升了 GMV。我们希望通过阿里云平台把 Blink 实时计算能力输出给阿里之外的所有企业，让他们能从中获益。以上就是本次的分享。

阿里小蜜这一年：从点到面的变迁

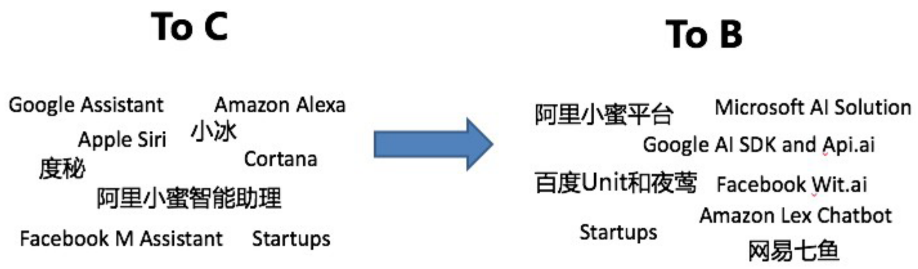
海青

一、生态领域的思考与阿里小蜜平台

1. 生态领域的思考

随着人工智能在全球领域的持续高涨，chatbot 人机交互作为其中一个分支在智能助理、智能服务、IOT 等领域进了白热化竞争态势，从全球大公司到创业公司纷纷加入战场并在一些独特的垂直领域开始精耕细作。在近两年的人机交互领域的发展中，一方面在 To C 端面向各个入口领域的竞争更加激烈（例如：在 IOT 领域的智能音箱）、垂直领域场景更加细分与丰富。另外一方面由 To C 市场的竞争开始转向 To B 市场的竞争，Google、Microsoft、Facebook、Amazon、百度、网易以及众多 startups 纷纷在 To B 领域通过 IaaS、PaaS 或者 SaaS 能力开始布局，并且基本围绕着如下两个体系进行输出：

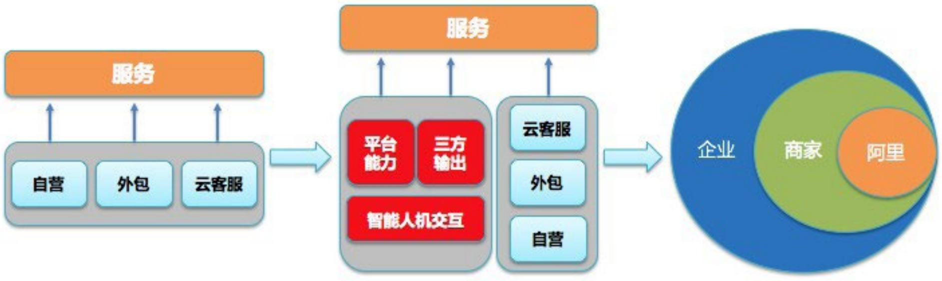
- 围绕着 IM 生态体系的绑定输出。例如：Facebook 在 Messager 平台上的 wit.ai 机器人平台输出、腾讯在微信体端也在逐步开发自己的机器人平台等
- 面向企业或者开发者领域的独立三方平台输出。例如：Amazon 的 Lex 平台、微软的 AI Solution、Google 的 AI SDK 与 Api.ai、网易七鱼等等



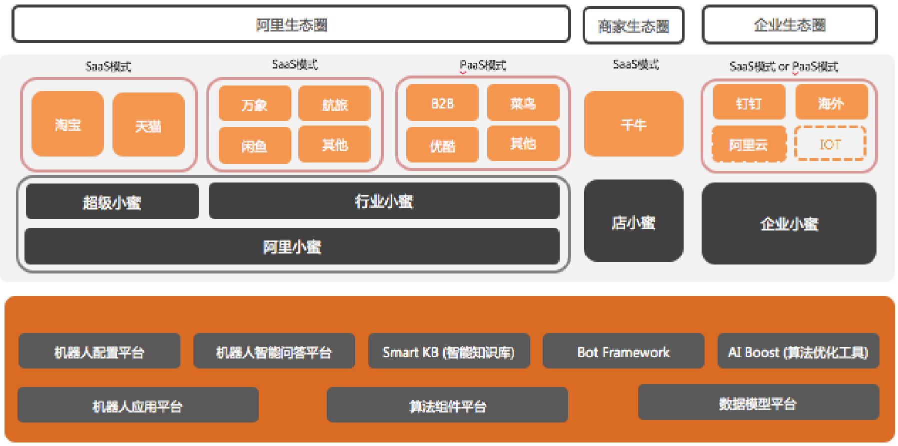
2. 面向生态的阿里小蜜平台

在人机交互行业生态领域发展的大背景下，面向智能服务领域，阿里小蜜在这两年时间同样在不断的升级与变迁：

- 在 To C 端，持续在智能 + 人的混合模式上逐步升级，并且把传统的服务往更大的泛服务领域扩展，除了传统服务的智能化、还拓展了智能导购服务、智能助理服务，闲聊服务等，以服务为基础向平台与多领域升级
- 在 To B 端，阿里小蜜从淘宝到阿里行业生态、二环商家生态，以及三环企业平台逐步拓展



最终形成了整个阿里小蜜平台面向阿里生态圈、商家生态圈、企业生态圈的整个体系大图：



- 面对消费者：超级小蜜在淘宝生态体系面向服务、导购、助理以及闲聊 4 个领域逐步进行深入的探索
- 面对阿里行业生态圈：通过平台化能力赋能阿里集团生态超过 30 个 BU，包含 AE、ICBU、1688、菜鸟、飞猪、闲鱼、淘票票、阿里内外等
- 面对商家生态圈：与千牛平台团队协同，基于 IM 消息体系，构建店小蜜体系，提供给商家全自动 + 半自动人工辅助智能对话能力
- 面对企业生态圈：依托于钉钉企业沟通生态圈的 IM 消息体系以及阿里云上的整个企业开放生态，企业小蜜和云小蜜也踏上了企业赋能之路

在过去的 2017 年阿里小蜜从阿里走向行业，逐步赋能商家和企业；从中国开始走向世界，覆盖英语、葡语、西班牙、印尼语、泰语，赋能 AE 及 Lazada 海外业务；从 PC、无线走向了 PC、无线和热线，在多端进行赋能；阿里小蜜全面从智能人机交互走向智能人机协同。

- 在过去的 2017 年阿里小蜜全年服务 3.4 亿名淘宝消费者，其中双 11 当天接待人次 904 万，智能服务占比达到 95%，智能服务解决率达到 93.1%
- 在过去的 2017 年赋能商家的店小蜜授权开启商家数达到 30w，其中双 11 当天机器人对话量超过 1 亿
- 在过去的 2017 钉钉端赋能企业数超过 1 万家；2017 年 10 月云栖大会正式开放云小蜜，截止到目前赋能 Lazada 东南亚服务业务，并逐步在多个行业领域与多家大中型企业推进云化服务体系

二、小蜜技术这一年

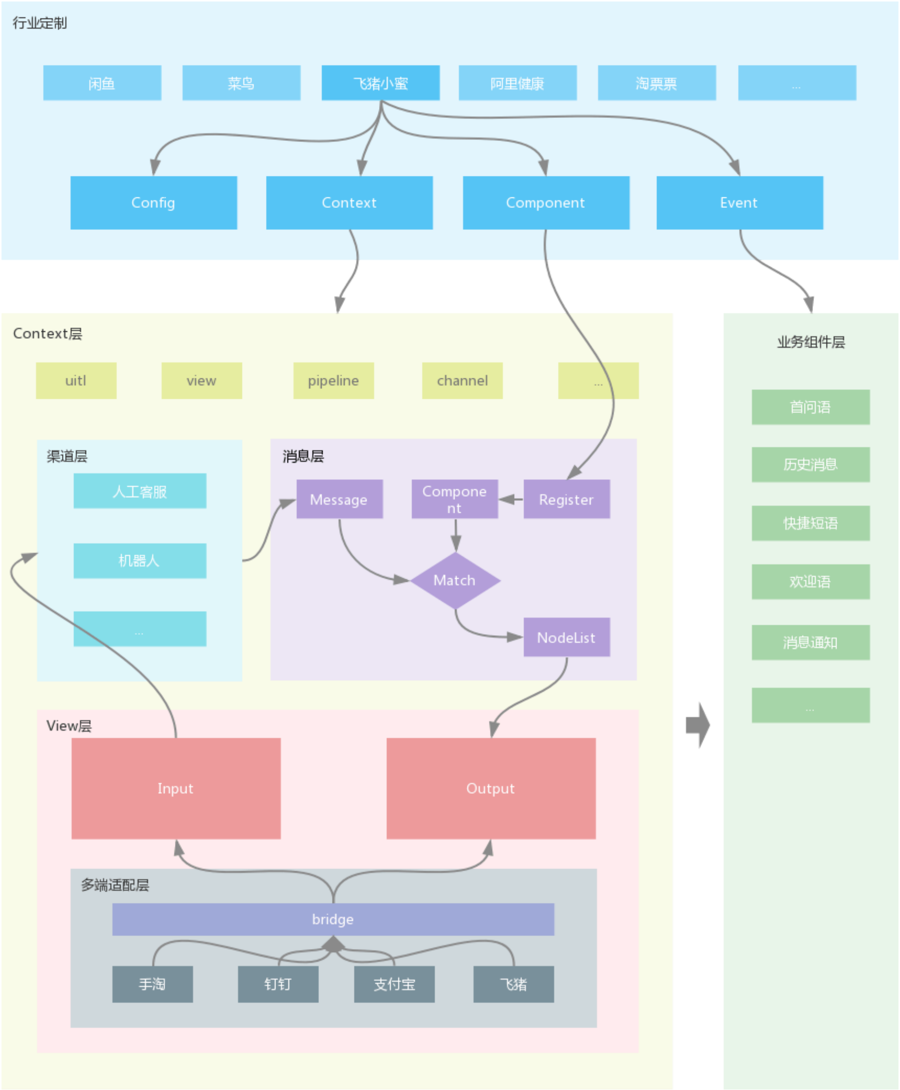
在技术这一端，过去的一年围绕着平台化和领域化不断完善和增强整个阿里小蜜平台：

- 架构体系端：围绕着 SaaS 和 PaaS 体系，逐步将前端和后端体系进行模块化处理，逐步完善和构建整个平台体系
- 算法端：逐步在单个领域结合实际业务场景进行纵深探索，并且不断进行创新

1. 前后端架构体系

前端架构体系

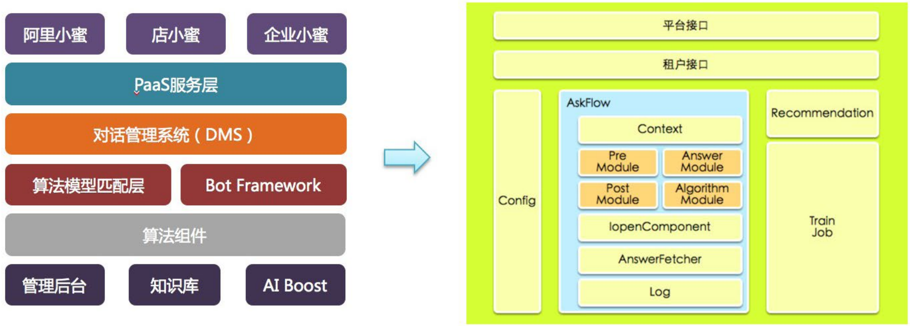
为了快速响应业务以及快速接入其他业务场景，阿里小蜜平台在前端架构上使用的是 WebApp 的技术选型。虽然 WebApp 在体验和功能上略逊于 Native App，但在快速响应业务需求和快速接入其他 APP 两方面相对 Native 优势明显。在经过 3 个大版本的不断升级与改造后，形成了按照模块划分的 7 层前端架构体系，如下图：



- 行业定制层：不同行业接入阿里小蜜平台，在既定的开放规范下对 UI、业务、插件进行定制的模块
- context 层：类似 koa.js/express.js 的设计，将 view/util/request/pipeline/channel 等核心模块挂载到 this 对象上，消息组件、业务模块等都执行在 this 这个 context 中。我们可以轻松对 this 进行扩展，以满足不同业务对于小蜜平台的特殊定制需求
- 渠道层：用户的输入在不同场景会发送给不同的处理对象，我们把处理对象称之为”渠道“，虽然目前的渠道只有机器人和人工客服，我们在架构上支持任意多个渠道，并且可以轻松切换对话渠道
- 消息层：将“对话”过程中的特定功能都抽象为具有特定消息类型的组件，这些组件在不同的渠道中都可以复用，阿里小蜜平台 90% 的功能扩展都可以通过消息层轻松实现
- 业务组件层：将”非对话层“的业务都抽象到这层中，提供串行加载、扩展等功能
- View 层：具有典型 WebIM 功能，提供 Input、output、addPlugin 等 API
- 客户端定制层：适配不同行业业务在不同的 APP 端的定制层

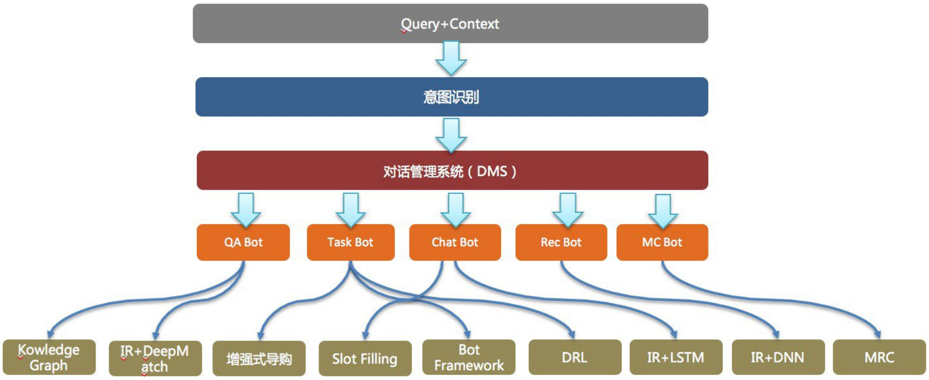
后端架构体系

- 面向整个阿里小蜜平台平台化，面向阿里生态圈、商家生态圈和企业生态圈支持以 PaaS 和 SaaS 输出
- 模块化整个对话管理和流程模块化，构建算法和业务模块可插拔的并行架构体系

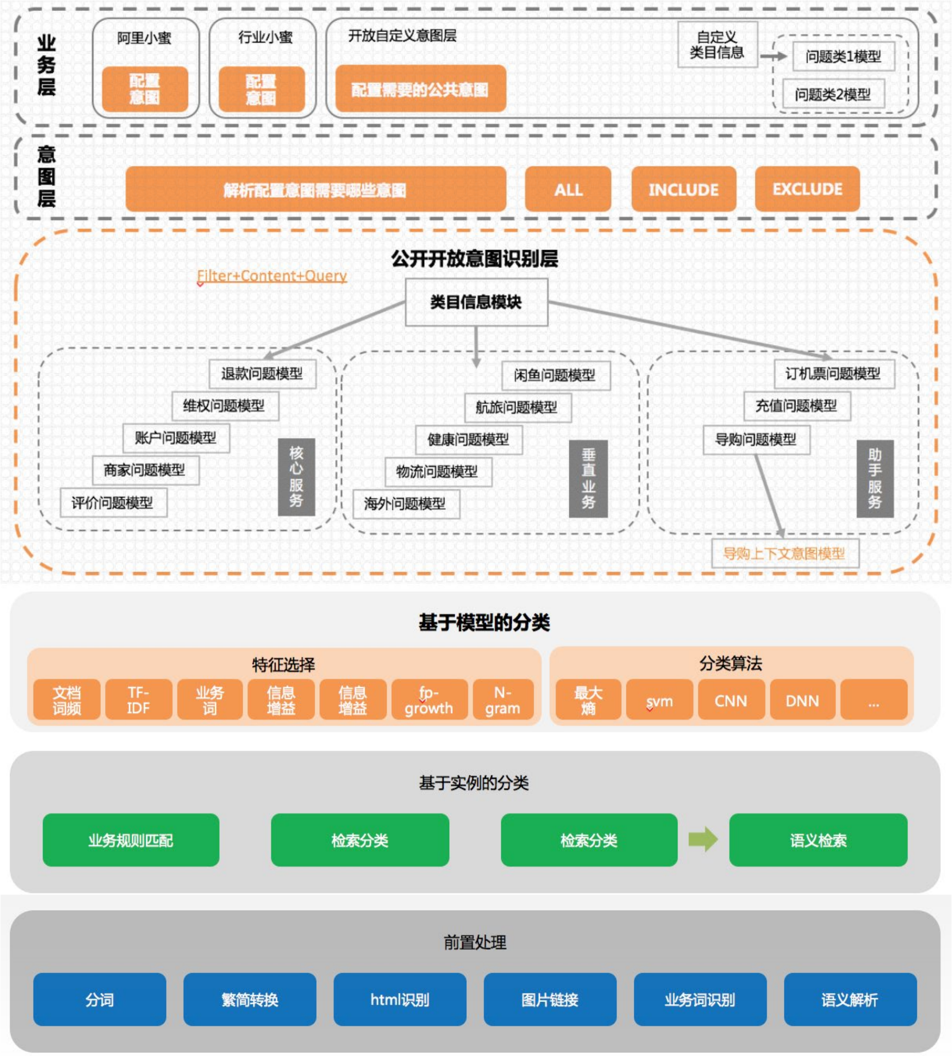


2. 算法体系

在算法体系持续按照面向不同的场景优化和升级整个算法体系模型，在 2017 年阿里小蜜平台的算法体系同样也按照领域化和平台化体系持续升级发展，整个人机交互架构如下：



- 意图识别：识别语言的真实意图，将意图进行分类并进行意图属性抽取。意图决定了后续的领域识别流程，因此意图层是一个结合上下文数据模型与领域数据模型不断对意图进行明确和推理的过程，完成意图的补全、意图分类和意图转移工作。整个意图识别按照模型可组合以及进行单独的算法选型



- 通过对话管理系统的控制，面向不同的领域场景采用不同的领域技术：
    - QA Bot：通过知识图谱、传统 IR 以及 DeepMatch 的方法完成知识问答的匹配
    - Task Bot：面向多领域技术完成任务型对话构建与问答
    - Chat Bot：完成闲聊机器人的问答
    - Rec Bot：完成推荐机器人的问答体系构建

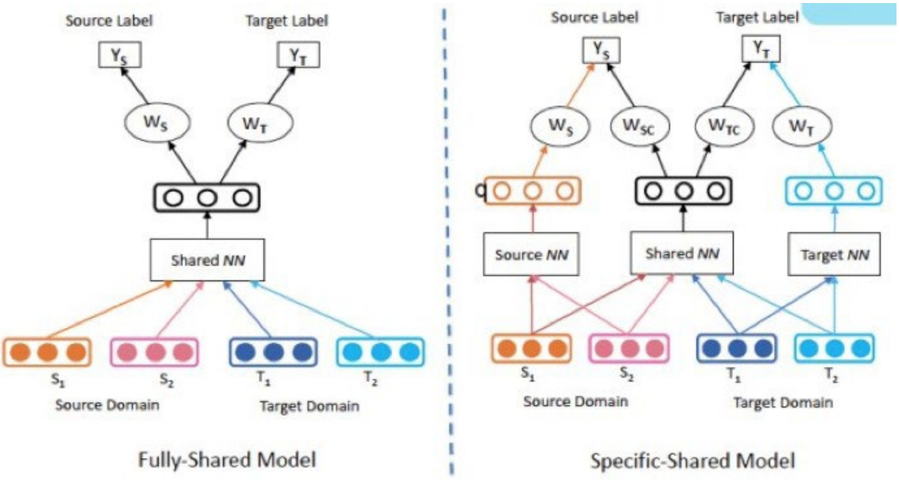
- MC Bot：在文档无法结构化的场景下（例如淘宝或者商家的活动场景），通过 Machine Reading 的方法来完成问答

下面选择几个比较有意思的工作做一下展开。

迁移学习下的 DeepQA 探索

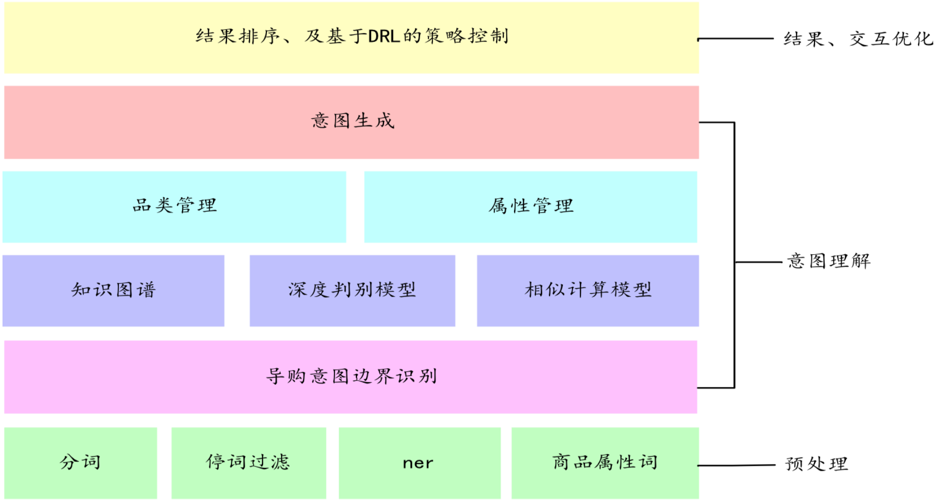
随着阿里小蜜平台不断的扩展，不仅需要在面向 C 端的手淘上帮助用户解决服务咨询类问题，而且也需要在新领域和国际化中承担起任务，而这些领域中存在标注数据量不足或者难以获得的问题（例如在 AliExpress 中的西班牙语场景）。基于此，我们希望利用已有的标注数据来优化小数据领域的 QA 匹配效果，而迁移学习在这方面就能发挥重要的作用。它最核心的思想就是将从一个环境中学到的知识用来帮助新环境中的学习任务。因此，在该场景下我们提出 DRSS-Adv 的迁移学习模型，用来建模 QA 匹配问题。

通常来说 ,TL 的模型有两类，一种是 unsupervised，另外一种是 supervised。前者假设完全没有目标领域的标注数据，后者假设仅有少部分目标领域的标注数据。具体大家可以看下文献 [2] 的综述。在我们的业务中主要关注 supervised 的迁移学习技术，同时结合深度神经网络（DNN）。在这个设定下主要有两种框架，一个是 Fully-shared(FS)，另外一个是 Specific-shared(SS)，框架图如下：



基于增强学习的智能导购

智能导购主要通过支持和用户的多轮交互，不断的理解和明确用户的意图。并在此基础上利用深度强化学习不断的优化导购的交互过程。下图展示了智能导购的技术架构图：



这里两个核心的问题：

a)：在多轮交互中理解用户的意图。

b)：根据用户的意图结果，优化排序的结果和交互的过程。

下面主要介绍导购意图理解、以及深度增强学习的交互策略优化。

智能导购的意图理解和意图管理：

智能导购下的意图理解主要是识别用户想要购买的商品以及商品对应的属性，相对于传统的意图理解，也带来了几个新的挑战：

- 用户偏向于短句的表达。因此，识别用户的意图，要结合用户的多轮会话和意图的边界。
- 在多轮交互中用户会不断的添加或修改意图的子意图，需要维护一份当前识别的意图集合。

- 商品意图之间存在着互斥，相似，上下位等关系。不同的关系对应的意图管理也不同。
- 属性意图存在着归类和互斥的问题。

针对短语表达，我们通过品类管理和属性管理维护了一个意图堆，从而较好的解决了短语表示，意图边界和具体的意图切换和修改逻辑。同时，针对较大的商品库问题，我们采用知识图谱结合语义索引的方式，使得商品的识别变得非常高效。下面我们分别介绍下品类管理和属性管理。

基于知识图谱和语义索引的品类管理

智能导购场景下的品类管理分为品类识别，以及品类的关系计算。下图是品类关系的架构图：

品类识别：

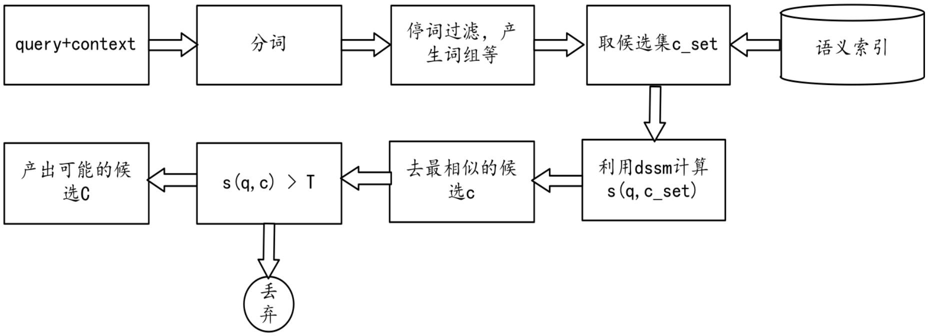
采用了基于知识图谱的识别方案和基于语义索引及 dssm 的判别模型。

a)：基于商品知识图谱的识别方案：

基于知识图谱复杂的结构化能力，做商品的类目识别。是我们商品识别的基础。

b)：基于语义索引及 dssm 商品识别模型的方案：

知识图谱的识别方案的优势是在于准确率高，但是不能覆盖所有的 case。因此，我们提出了一种基于语义索引和 dssm 结合的商品识别方案兜底。



语义索引的构造：

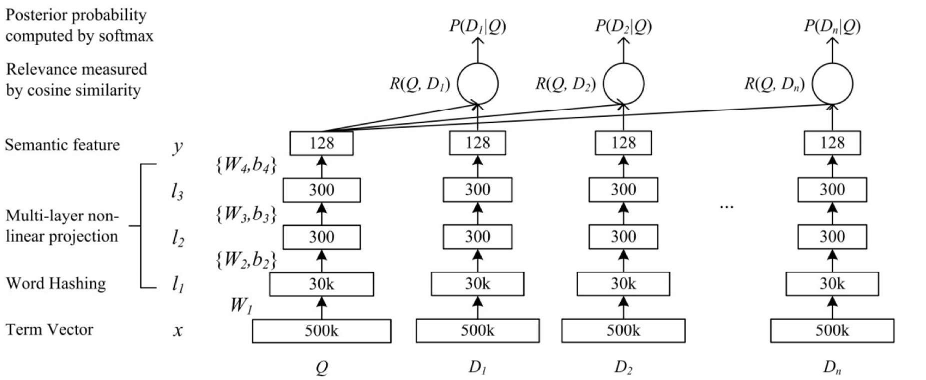
通常语义索引的构造有基于本体的方式，基于 LSI 的方式。我们用了一种结合搜索点击数据和词向量的方式构造的语义索引。主要包括下面几步：

- 利用搜索点击行为，提取分词到类目的候选。
- 基于词向量，计算分词和候选类目的相似性，对索引重排序。

基于 dssm 的商品识别：

dssm 是微软提出的一种用于 query 和 doc 匹配的有监督的深度语义匹配网络，能够较好的解决词汇鸿沟的问题，捕捉句子的内在语义。本文以 dssm

作为基础，构建了 query 和候选的类目的相似度计算模型。取得了较好的效果，模型的 acc 在测试集上有 92% 左右。



样本的构造：训练的正样本是通过搜索日志中的搜索 query 和点击类目构造的。负样本则是通过利用 query 和点击的类目作为种子，检索出来一些相

似的类目，将不在正样本中的类目作为负样本。正负样本的比例 1 ： 1。

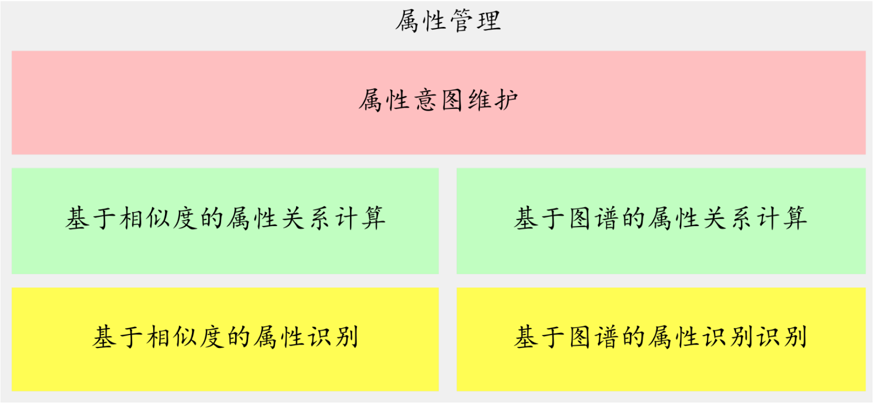
品类关系计算：

品类关系的计算主要用于智能导购的意图管理中，这里主要考虑的几种关系是：上下位关系和相似关系。举个例子，用户的第一个意图是要买衣服，当后面的意图说要买水杯的时候，之前衣服所带有的属性就不应该被继承给水杯。相反，如果这个时候用户说的是要裤子，由于裤子是衣服的下位词，则之前在衣服上的属性就应该被继承下来。

- 上下位关系的计算 2 种方案：
  - 采用基于知识图谱的关系运算。
  - 通过用户的搜索 query 的提取。
- 相似性计算的两种方案：
  - 基于相同的上位词。比方说小米，华为的上位词都是手机，则他们相似。
  - 基于 fast-text 的品类词的 embedding 的语义相似度。

基于知识图谱和相似度计算的属性管理

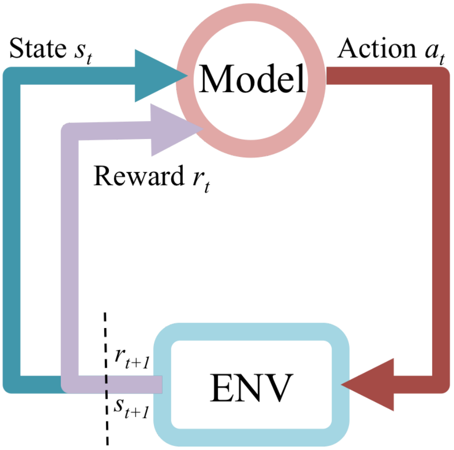
下图是属性管理的架构图：



整体上属性管理包括属性识别和属性关系计算两个核心模块，思路和品类管理较为相似。这里就不在详细介绍了。

深度强化学习的探索及尝试

强化学习是 agent 从环境到行为的映射学习，目标是奖励信号（强化信号）函数值最大，由环境提供强化信号评价产生动作的好坏。agent 通过不断的探索外部的环境，来得到一个最优的决策策略，适合于序列决策问题。下图是一个强化学习的 model 和环境交互的展示：



深度强化学习是结合了深度学习的强化学习，主要利用深度学习强大的非线性表达能力，来表示 agent 面对的 state 和 state 上决策逻辑。目前我们用 DRL 主要来优化我们的交互策略。因此，我们的设定是，用户是强化学习中的 env，而机器是 model。action 是本轮是否出主动反问的交互，还是直接出搜索结果。

状态 (state) 的设计：

这里状态的设计主要考虑，用户的多轮意图、用户的人群划分、以及每一轮交互的产品的信息作为当前的机器感知到的状态。

state =（intent1, query1, price1, is_click, query_item_sim, …, power, user_inter, age)

其中 intent1 表明的是用户当前的意图，query1 表示的用户的原始 query。

price1 表示当前展现给用户的商品的均价，is_click 表示本轮交互是否发生点击，query_item_sim 表示 query 和 item 的相似度。power 表示是用户的购买力，user_inter 表示用户的兴趣 , age 表示用户的年龄。

reward 的设计：

由于最终衡量的是用户的成交和点击率和对话的轮数。因此 reward 的设计主要包括下面 3 个方面：

a)：用户的点击的 reward 设置成 1

b)：成交设置成 [1 + math.log(price + 1.0)]

c)：其余的设置成 0.1

DRL 的方案的选型：这里具体的方案，主要采用了 DQN，policy-gradient 和 A3C 的三种方案。

基于可配置 Bot Framework 体系构建

在阿里小蜜平台中还存在一种需要动态获取系统数据并且整个流程相对较为个性化的场景，这种场景的体系数据偏少甚至没有并且需要跟对应的 ERP 等系统打通，因此我们就构建以一套 Bot Framework 系统，来满足对应企业的运营或者开发同学完成个性化任务体系的定制。

整个 BFW1.0 的体系支持：

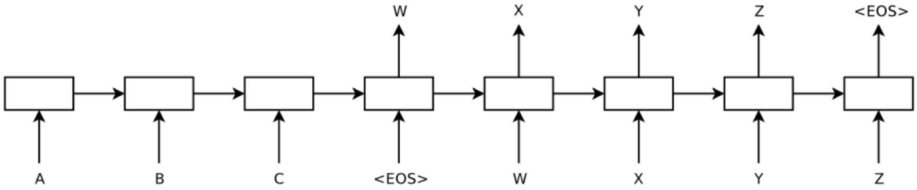
- 自定义多轮对话流程
- 自定义意图、实体以及实体值
- 支持第三方接口在多轮交互平台的接入

尽管如此，整个 BFW1.0 面向复杂业务以及面向开发者的灵活度不足，因此在 2018 年，我们将 BFW1.0 按照 chatflow 的体系思路进行升级成为 BFW2.0。

基于检索模型和深度学习模型相结合的聊天应用

智能聊天的特点：非面向目标，语义意图不明确，通常期待的是语义相关性和渐进性，对准确率要求相对较低。面向 open domain 的聊天机器人目前无论在学术界还是在工业界都是一大难题，通常在目前这个阶段我们有两种方式来做对话设计：

一种是学术界非常火爆的 Deep Learning 生成模型方式，通过 Encoder-Decoder 模型通过 LSTM 的方式进行 Sequence to Sequence 生成，如下图：



Generation Model(生成模型)：

优点：通过深层语义方式进行答案生成，答案不受语料库规模限制

缺点：模型的可解释性不强，且难以保证一致性和合理性回答

另外一种方式就是通过传统的检索模型的方式来构建语聊的问答匹配。

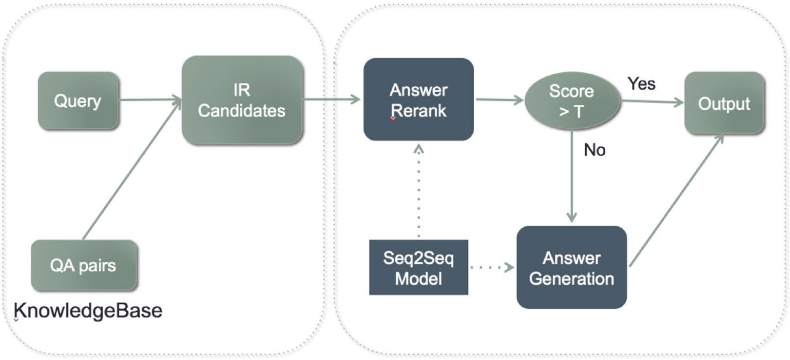
Retrieval Model(检索模型)：

优点：答案在预设的语料库中，可控，匹配模型相对简单，可解释性强

缺点：在一定程度上缺乏一些语义性，且有固定语料库的局限性

因此在阿里小蜜的聊天引擎中，我们结合了两者各自的优势，将两个模型进行了融合形成了阿里小蜜聊天引擎的核心。先通过传统的检索模型检索出

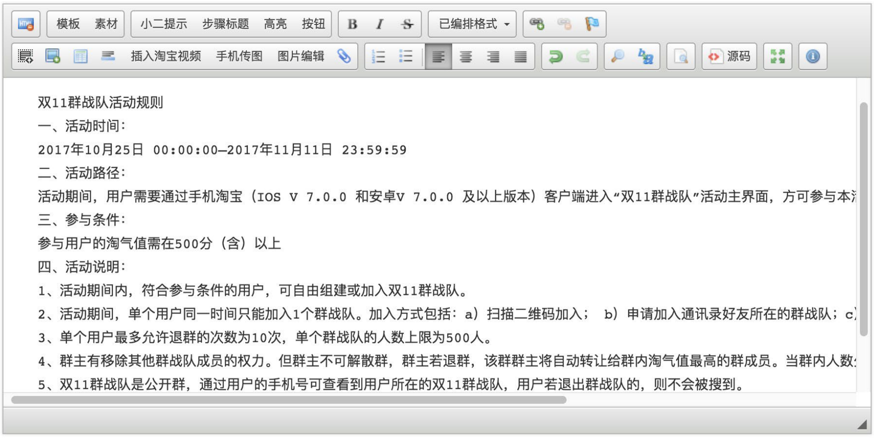
候选集数据，然后通过 Seq2Seq Model 对候选集进行 Rerank，重排序后超过制定的阈值就进行输出，不到阈值就通过 Seq2Seq Model 进行答案生成，整体流程图如下：



机器阅读理解技术探索与实践

在阿里小蜜平台的业务体系中，存在大量知识数据是无法通过先验知识结构化或者结构化效率极低的场景，例如淘宝双十一大促的活动、税务法律等等。因此我们通过机器阅读理解的运用，可以减少人工知识点拆解工作，让机器直接对规则进行阅读，为用户提供规则解读服务，是最自然的交互方式。因此在 2017 年阿里巴巴 iDST 团队与我们阿里小蜜团队合作共同在机器阅读领域进行了深入的探索和应用。

例如，如下图是 2017 年淘宝双十一“群战队”一个活动的规则：



双11群战队活动规则
一、活动时间：
2017年10月25日 00:00:00—2017年11月11日 23:59:59
二、活动路径：
活动期间，用户需要通过手机淘宝（IOS V 7.0.0 和安卓V 7.0.0 及以上版本）客户端进入“双11群战队”活动主界面，方可参与本次
三、参与条件：
参与用户的淘气值需在500分（含）以上
四、活动说明：
1、活动期间内，符合参与条件的用户，可自由组建或加入双11群战队。
2、活动期间，单个用户同一时间只能加入1个群战队。加入方式包括：a）扫描二维码加入； b）申请加入通讯录好友所在的群战队；c）
3、单个用户最多允许退群的次数为10次，单个群战队的人数上限为500人。
4、群主有移除其他群战队成员的权力。但群主不可解散群，群主若退群，该群群主将自动转让给群内淘气值最高的群成员。当群内人数
5、双11群战队是公开群，通过用户的手机号可查看到用户所在的双11群战队，用户若退出群战队的，则不会被搜到。

运营同学在知识库中录入群战队的活动规则文本，阿里小蜜即可直接基于文本内容的阅读来提供问答服务

机器阅读理解技术机器阅读理解模型已经在学术界取得了相当大的突破，但由于学术界和工业界具体场景的巨大差异，要将机器阅读理解技术运用到实际业务场景中，还存在相当大到挑战。

- 设计具有针对性的业务模型

  从上面的例子可以看到，业务场景中的活动规则、法规文档具有一定的独特性，往往包含大量文档结构，如大小标题和文档的层级结构等。此外，不仅文章普遍比较长，答案也较长，往往跨多个句子。这与一些公开模型所阅读的 wiki 百科类数据特点有巨大的差异。需要针对场景的特点来设计模型结构。

- 模型的性能考虑

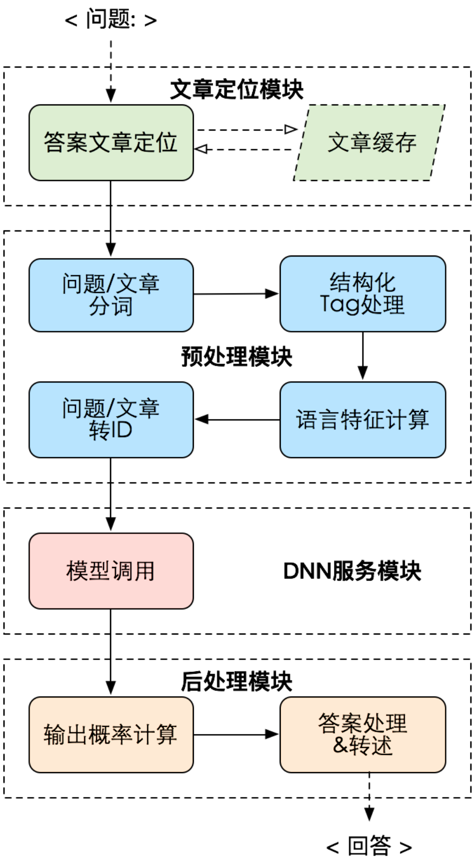
  学术成果中的模型一般都较为复杂，而工业界场景中由于对性能的要求，无法将这些模型直接在线上使用，需要做一些针对性的简化，使得模型效果下降可控的情况下，尽可能提升线上预测性能。

- 模型领域的迁移

  目前的机器阅读模型是领域相关的，这使得领域的快速拓展成为一大挑战，阿

里小蜜的活动规则解读是我们支持的第一个场景，目前我们已经拓展了税务法规解读场景，未来还会在金融、电信等领域进行开拓，因此在新领域拓展时，需要利用到过去学习过的知识。

基于机器阅读理解模型的在线问答流程如下图所示，在解析用户问题到输出答案的过程中，共有 4 个模块参与处理：



1. 文章片段定位模块

文章片段定位模块针对用户问题，召回候选的文档段落集合供机器阅读产生答案。借助该模块，可帮助缩减机器阅读理解模型的计算量，同时在一定程度上提升准确率。例如用户提问「天猫造物节的活动什么时候开始?」，该模块需要返回所有与

「天猫造物节」有关的文档段落。定位的方式可以通过文本分类、文本检索或者问题模板拦截来完成，文本分类需要提前标注数据训练模型，目前我们的流程中主要以无监督的文本检索或者人工维护的问题模板定位为主。

2. 预处理模块

预处理模块包含 4 个步骤，首先需要对用户问题和待阅读文章做文本分词，由于带阅读文章常常伴有多级段落、特殊标签等结构信息，需要做格式化处理。

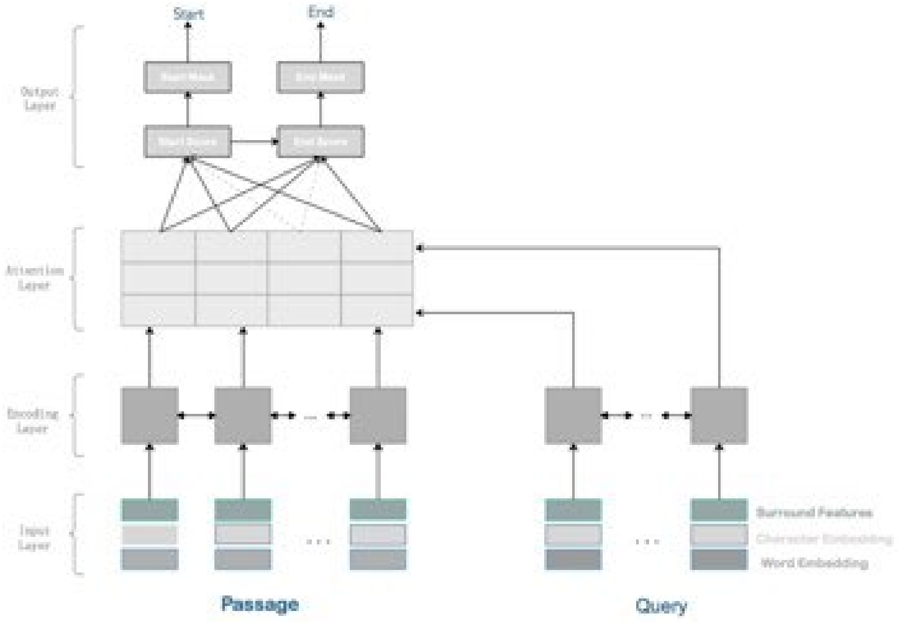
3. 在线服务模块

DNN 在线服务模块部署了本文前半部分描述的深度机器阅读理解模型，在接受第 2 步处理好的 <question, doc> 向量后，计算输出文章中的词语或者句子作为答案的概率。

4. 后处理模块

后处理模块负责具体答案的构建，根据在线服务模块输出的答案概率，其按照特定策略计算出最可能的答案

模型结构如下图：



就在最近我们机器阅读模型已排到 SQuAD 榜的 top1，且 ExactMatch 指标首次超越了人类表现。

Leaderboard

Since the release of our dataset, the community has made rapid progress! Here are the ExactMatch (EM) and F1 scores of the best models evaluated on the test and development sets of v1.1. Will your model outperform humans on the QA task?

Rank	Model	EM	F1
1 Jan 05, 2018	SLQA+ (ensemble) *Alibaba iDST NLP*	82.440	88.607
2 Dec 22, 2017	AttentionReader+ (ensemble) *Tencent DPDAC NLP*	81.790	88.163
2 Dec 17, 2017	r-net (ensemble) *Microsoft Research Asia* http://aka.ms/rnet	82.136	88.126
3 Nov 17, 2017	BiDAF + Self Attention + ELMo (ensemble) *Allen Institute for Artificial Intelligence*	81.003	87.432
4 Dec 28, 2017	SLQA+ (single model) *Alibaba iDST NLP*	79.199	86.590
4 Jan 03, 2018	r-net+ (single model) *Microsoft Research Asia*	79.901	86.536
5 Dec 05, 2017	SAN (ensemble model) *Microsoft Business AI Solutions Team* https://arxiv.org/pdf/1712.03556.pdf	79.608	86.496
6 Oct 17, 2017	Interactive AoA Reader+ (ensemble) *Joint Laboratory of HIT and iFLYTEK*	79.083	86.450
7 Oct 24, 2017	FusionNet (ensemble) *Microsoft Business AI Solutions Team* https://arxiv.org/abs/1711.07341	78.978	86.016

三、未来的展望与挑战

- 交互式智能的也逐渐成为了新的领域入口，在未来需要基于场景数据的基础上，需要持续结合技术、产品进行面领域性场景性探索与积累；需要持续在领

域数据和知识体系的持续积累；在技术领域，我们将持续在生成模型、增强学习、迁移学习、机器阅读、情感化等持续深入。

- 在未来阿里小蜜平台会持续在平台化和垂直领域方向持续深入下去，围绕着行业、商家、企业以及整个 chatbot 生态构建智能服务的阿里小蜜智能服务平台。

References

[1] Feng-Lin Li, Minghui Qiu, Haiqing Chen, Xiongwei Wang, Xing Gao, Jun Huang, Juwei Ren, Zhongzhou Zhao, Weipeng Zhao, Lei Wang, Guwei Jin, Wei Chu: AliMe Assist : An Intelligent Assistant for Creating an Innovative E-commerce Experience. CIKM 2017: 2495-2498

[2] Jianfei Yu, Minghui Qiu, Jing Jiang, Shuangyong Song, Jun Huang, Wei Chu and Haiqing Chen, et al. Modelling Domain Relationships for Transfer Learning on Retrieval-based Question Answering Systems in E-commerce[C]// WSDM 2018.

[3] Yin W, Schütze H, Xiang B, et al. ABCNN: Attention-Based Convolutional Neural Network for Modeling Sentence Pairs[J]. Computer Science, 2015.

[4] Hu B, Lu Z, Li H, et al. Convolutional neural network architectures for matching natural language sentences[J]. 2015, 3:2042-2050.

[5] Pang L, Lan Y, Guo J, et al. Text Matching as Image Recognition[J]. 2016.

[6] Sukhbaatar S, Szlam A, Weston J, et al. End-To-End Memory Networks[J]. Computer Science, 2015.

[7] Wu Y, Wu W, Xing C, et al. Sequential Matching Network: A New Architecture for Multi-turn Response Selection in Retrieval-Based Chatbots[C]// Meeting of the Association for Computational Linguistics. 2017:496-505.

[8] Huang P S, He X, Gao J, et al. Learning deep structured semantic models for web search using clickthrough data[C]// ACM International Conference on Conference on Information & Knowledge Management. ACM, 2013:2333-2338.

[9] Pengfei Liu, Xipeng Qiu, and Xuanjing Huang. 2017. Adversarial Multi-task Learning for Text Classification. In ACL.

菜鸟仓配自动化 UCS 揭秘

兮扬

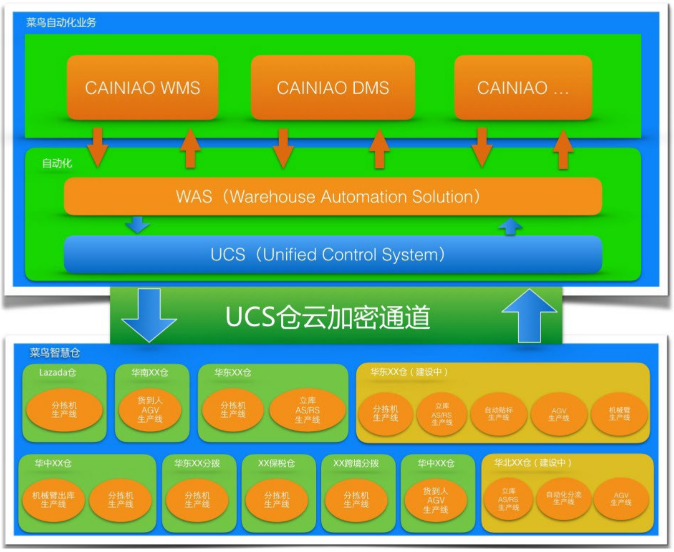
自动化背景介绍

菜鸟正在持续增长，17 年双十一菜鸟配送物流订单已经到达 8.12 亿，相信 10 亿订单包裹很快即可实现。劳动力将成为整个物流行业的一个瓶颈，劳动力持续减少，菜鸟物流包裹在迅猛增涨。在这两个矛盾愈发明显的情况下，用自动化生产流水线来替代人来做一部分简单重复性的工作势在必行。菜鸟自动化也就是在这种环境下诞生的。

一、什么是 UCS

UCS 是 Unified Control System 简称，UCS 成立的初衷是用一套控制系统实现对菜鸟下的所有自动化设备进行调度控制，让菜鸟上游的业务系统屏蔽掉底层所有设备的差异，统一走 UCS 实现对整个菜鸟仓、分拨中心自动化设备的调度控制。

二、UCS 在菜鸟仓配体系下的位置



1. UCS 在整个菜鸟仓配体系下，是一个集中的生产线控制系统，它处于业务系统与自动化生产线之间，对业务系统暴露自动化任务接口，对下控制整个菜鸟智慧仓生产线，调度生产线进行生产，同时接收生产线上报上来的生产数据。

三、UCS 与 WCS 之间的关系

WCS

1. WCS 即 Warehouse Control System，它是一个纯仓库设备的控制系统，一般来讲一个 WCS 控制一套自动化生产线。
2. WCS 通常部署在仓库内。
3. WCS 通常与自动化设备一块交付的。
4. 一般来讲 WMS 直接对接 WCS。

UCS

1. UCS 是一个自动化设备生产线控制系统，UCS 的目标是控制所有菜鸟仓储域下的自动化设备生产线。
2. UCS 是菜鸟统一的 WCS，它不是为某一条生产线定制的，它对下控制菜鸟所有智慧仓的自动化设备，对上层业务提供一套统一的 API。用于各个业务对自动化生产线下任务，从而驱动仓内的自动化生产线执行生产任务，进而完成仓内的生产作业。
3. 因为有了 UCS，上游的业务系统，不再需要关心具体自动化生产线的 WCS，只需要对接 UCS，即可完成自动化业务的接入。
4. UCS 从职责上来说，它是一个统一的 WCS，UCS 可以对接 WCS，WCS 不可取代 UCS。

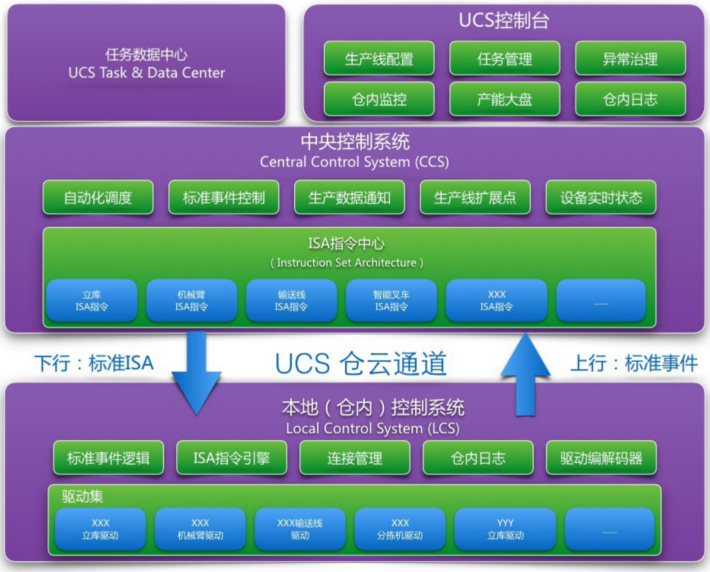
四、UCS 需要解决的问题

1. 解决菜鸟仓配下各种自动化生产线的集成接入，目前接入 UCS 的生产线有：货到人 AGV，分拣机，立库（AS/RS），机械臂等。
2. 对业务屏蔽下游各种异构厂商生产线，统一对上游业务的接口。比如立库

有新松、昆船、中鼎、大福等。分拣机有：德马泰克、范德兰德、德玛、Swisslog、力生等，不同的厂家有不同的通讯协议，对上游来说这些差异都不用感知。

3. 解决自动化生产线快速复制能力，比如一条自动化生产线，需要快速的复制到十个仓，需要一套标准化的接入体系，便于自动化业务的快速接入。
4. 解决路由的控制指令通道的高可用，快速的送达，确保在弱网的情况下，指令也可以做到下发，确保生产的有序。
5. 解决复杂生产线硬件设备强调度协作的问题，比如立库，出入库可能都涉及到输送线、堆垛机、RGV 这些设备之间需要协作配合才能完成一个托盘出入库。
6. 解决高速有序安全的生产作业问题，比如分拣业务，1 小时要分拣 2W 单，而且顺序不能乱。
7. 解决自动化场景下的业务监控，做到优于业务提前发现问题，定位问题的能力，成为安全生产的有力保障。

五、UCS 架构



整个 UCS 架构主体上分为五个部分：

- TDC：任务数据中心
- UCS 控制台
- 中央控制系统（CCS）
- UCS 仓云通道
- 本地控制系统（LCS）

任务 & 数据中心（TDC）

一句话总结：UCS 自动化的核心数据中心，任务相关的数据以及结果都由 TDC 来存储。

1. 接受上游业务系统的自动化任务，同时接受与自动化任务相关业务数据，比如分拣机包裹数据、面单信息等。
2. 自动化执行过程中，一些中间上下文数据的存储。
3. 自动化执行完成后，自动化任务的结果数据存储。
4. 对上游业务提供一些任务结果数据查询的 API，用于业务上因为任务结果消息通知没收到后的业务补偿逻辑。
5. 对 UCS 内部提供自动化任务执行的数据支撑。
6. 对 UCS 控制台提供自动化任务信息、数据信息、调度信息等数据支撑，用于控制台渲染。

中心控制系统（CCS）

1. 中央控制系统的核心职责是，受理自动化任务：
    - 对某些调度类的自动化任务做调度路径规划，然后按调度的路径来顺序执行，每一段调度路径可以映射成一个 ISA 指令，然后将 ISA 指令经过 UCS 云仓通道下发到本地控制系统，交由本地控制系统来执行指令。对于非调度类的自动化任务，中央控制系统可能会将受理的自动化任务，通过云仓通道下发到仓内，再由本地控制系统来执行任务。

2. 通过标准事件控制，CCS 知道什么样的事件会触发什么样子的行为，比如一个立库托盘扫码的事件，对 CCS 来说就是尝试执行入库逻辑。
3. 通过自动化调度模块，调度整个自动化任务，协调自动化生产线上不同设备之间的协作顺序。
    - 以立库上架为例，托盘从上架口进去，先会经过输送线输送，输送到堆垛机码头后，堆垛机移动到堆垛机码头来取货，取完货堆垛机要执行送货的指令，从而完成整个上架过程，这个过程本身就是一个调度，什么时候通知输送线移动，什么时候通知堆垛机来取货，以及什么时候堆垛机执行送货，整个过程是连贯的，需要调度系统来指挥每个步骤什么时候来执行。
4. 它是一个指令中心，掌控所有生产线的 ISA，下发 ISA。ISA 即 UCS 标准控制指令，结合生产线，CCS 知道每一次 LCS 上来的事件，应该会触发什么样的 ISA 交互。CCS 能准确的将 ISA 下发到具体生产线的 LCS 上。
5. 通过生产线扩展点，可以完成针对不同生产线对应的一些标准事件处理逻辑的一些扩展，方便特殊生产线做一些扩展性的业务逻辑。
6. 通过设备实时状态，可以记录生产过程中，设备的一些实时状态，用于业务上的一些算法调度。
7. 通过生产数据通知，可以将仓内生产的一些事件通知上游业务系统。

本地控制系统（LCS）

1. 本地控制系统核心职责与自动化设备建立连接通讯，通过给设备发送指令的方式，让自动化生产线执行自动化作业任务。
2. 通过驱动适配不同的设备厂家协议，LCS 对驱动输入 ISA，然后驱动将 ISA 解析成厂商指令，然后发给厂商的 PLC 或者 WCS，从而让自动化生产线执行作业。待自动化生产线执行完后，驱动再将设备报上来的事件统一成标准事件，交给 LCS 统一处理。
3. 同时当 ISA 执行完毕后，LCS 需要走 UCS 云仓通道向上报 ISA 的执行结果。
4. LCS 还具备仓内执行日志，以及通信日志的查询接口，用于直接在云端拉取仓内日志，方便业务排查问题。

UCS 控制台

1. 它是一个生产线配置中心，通过它，可以完成一条生产线的开仓配置，初始化生产线。
2. 它是一个自动化任务的控制台，通过它，可以跟踪到自动化任务的状态。
3. 通过控制台还可以用于一些生产异常情况的处理，提供比如对任务强制成功，强制失败，对任务步骤重试等这些功能。方便现场自动化生产线的运营维护。
4. 监控，即对自动化作业进行有效的监控，可以优于业务提前发现问题，并且快速通知指导现场如何解决问题，避免生产线卡住，可以人在千里之外，运筹帷幄，对现场情况了如指掌。

写在最后

UCS 承担着菜鸟自动化控制链路上比较核心的一个环节。我们在过去一年的时间内，通过我们的架构支撑了若干个菜鸟自动化项目，未来一年有数倍规模的菜鸟智慧仓等着我们去支撑落地。2017 年是菜鸟的自动化元年，若干个菜鸟自动化场景完成了从 0 到 1 的过程，随着阿里集团未来 5 年 1000 亿的投资注入菜鸟，18 年菜鸟自动化将迎来爆发式的增长，有更多的自动化项目等着去支撑，更多复杂的自动化场景等着去落地。我们需要更多的有志之士和我们一块去完善我们的架构，去支撑菜鸟的自动化快速发展，去迎接更多有意思的架构挑战，准备好了么？加入我们吧。

阿里怎么发红包？自研智能权益系统首次公开

志昭

背景

每年双 11 大促的预热阶段和双 11 当天，消费者都可以在淘宝、天猫领到或者兑换到各种各样的用户权益，比如购物津贴、红包、店铺优惠券、品类券等等。对于平台来说，采用最优化的策略发放这些不同类型的用户权益，可以有效的平衡消费者、商家、平台这三方的利益：消费者可以用更加优惠的价格买到心仪的商品；商家多了一种让利的方式，消费者因为实惠产生了更多的购买，从而提升商家的销售额；平台丰富了大促的营销玩法和氛围，有利于整体 GMV 和客单价的提升。

业务分析

双 11 购物津贴是 2017 年大促重量级的优惠营销玩法，是天猫双 11 全球购物狂欢节全场通用购物抵用券，用于抵扣商品支付金额，购物津贴按类目维度阶梯设置满减门槛，可以在天猫、航旅等平台跨店凑单使用。购物津贴在消费者购物的过程中可以起到促进成交、提高客单价的作用。

红包是另一种常见的权益形式，具有宣传造势及社交的软效果，结合不同的发放或者领取方式，今年双 11 有火炬红包、狂欢城红包、切红包等多种互动玩法。其中火炬红包希望通过简单的红包获取方式、爆发式的红包传播、期权式的红包统一开奖，形成全民可玩的话题爆点，提升双 11 活动影响力。

不同类型的权益和玩法有着不同的营销目的，形成了不同的发放策略，对于发放节奏、金额的控制有多种因素需要考虑。我们以购物津贴和红包的发放为例，列举了几个需要解决的核心问题。

需要解决的问题

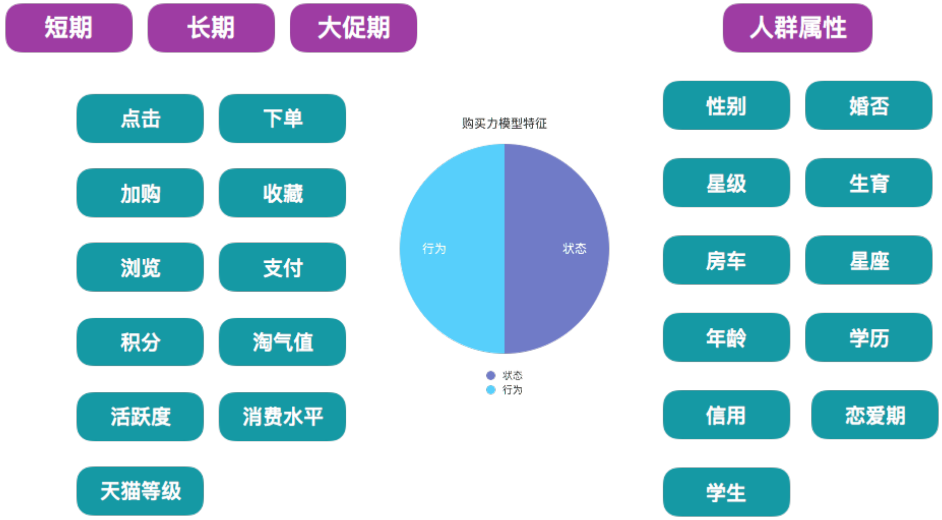
- 消费者双 11 当天消费金额预估—根据预估的消费金额，计算消费者需要的购物津贴，确保购物津贴有稀缺感的同时，最大化的促进成交
- 消费者优惠敏感度分析—对于不同的消费者，发放不同金额的红包，在保证使用率的同时，可以惠及更多的消费者。
- 消费者传播力分析—在火炬红包活动中，可以给高传播用户更高获取未点亮红包的概率，以触达、传播更多的用户。

- 消费者类目偏好分析、流失分析—用于产出需要重点关注的人群（比如“偏科”人群、流失人群、光看不买人群），可以发放特定的红包提升用户转化。

针对上述问题，产出了不同的模型。可以针对不同的营销目的，组合使用多种模型，形成相应权益类型的发放策略。以下将简要介绍各个模型及部分业务结果。

双 11 当天消费金额预估模型

根据消费者历年双 11 的购买记录，产出一个可以根据消费者近期行为数据预测 2017 年双 11 当天消费金额的模型。消费者的特征主要分为行为特征和属性特征，其中行为特征又分为短期、长期、大促期行为，如下图所示：

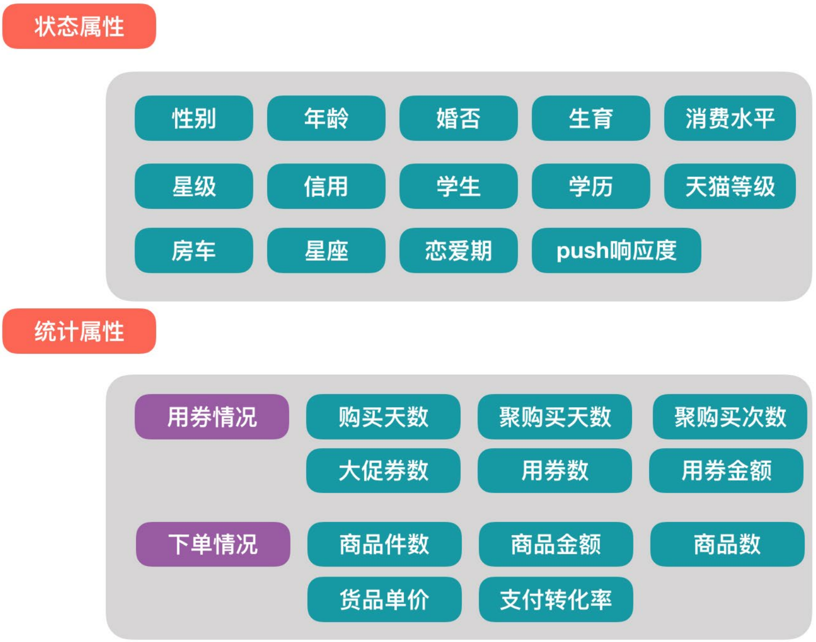


消费金额预估中比较核心的两个模型是消费金额回归模型和高额人群分类模型，分别用于解决一般用户的消费金额预估问题以及高额用户的判别问题。一般用户的消费金额预估模型对于高消费人群的预测结果偏差较大，使用分类模型对于区分高额人群的效果比较好：

通过准确的预测消费者双 11 当天的消费金额、简化使用方式、优化发放策略，使用购物津贴的消费者占比提升 51%，使用购物津贴的 GMV 占比提升 72%。

消费者优惠敏感度模型

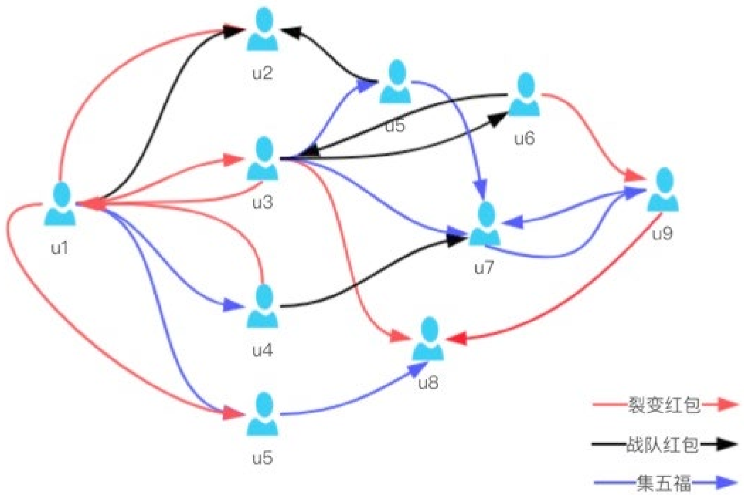
通过消费者使用各种权益的历史购买记录，分析消费者对不同优惠金额的敏感程度。选用 XGBoost 模型，使用了如下的用户特征：



线上 A/B 测试表明，在红包整体发放金额相同的条件下，按照消费者权益敏感度进行个性化发放，对于特定人群，红包人均使用率提升了 17.7%，人均消费金额提升 16.4%。

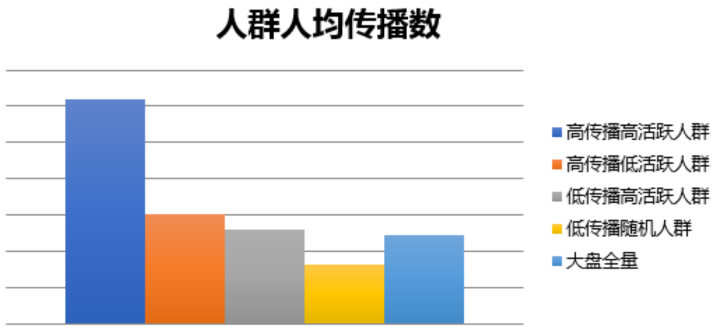
消费者传播力模型

通过分析消费者历史互动行为，利用互动、次数等特征拟合用户传播力因子；以近半年消费者平台行为为基础，利用登录、浏览、加购、收藏、购买等特征拟合用户活跃度因子。根据用户传播力因子和用户活跃度因子，将人群划分为高传播高活跃人群、高传播低活跃人群、低传播高活跃人群、低传播随机人群。



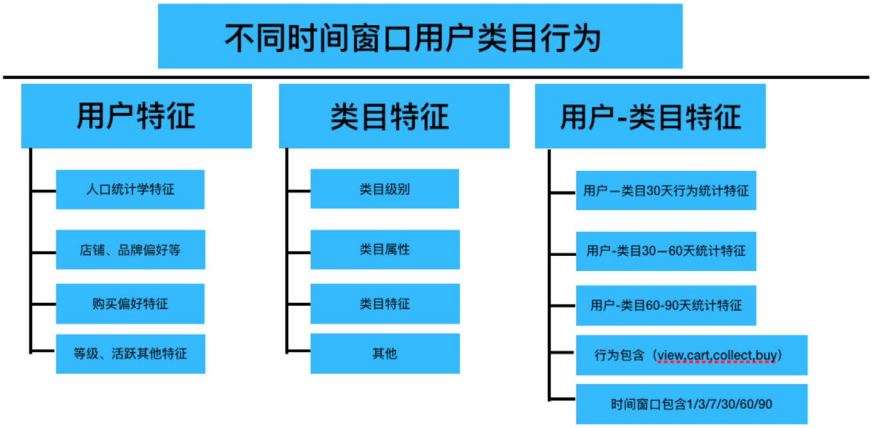
用户活动互动关系图

在火炬红包的传播过程中，高传播高活跃人群表现出了更强的传播能力。可以利用不同人群的传播特性，有针对性的优化火炬红包的玩法，使得活动的参与人数最大化，提升火炬红包的影响力。

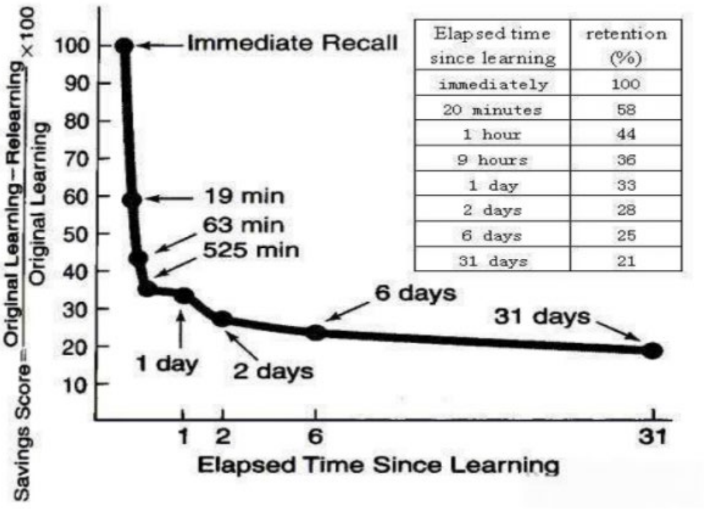


消费者类目偏好模型

消费者类目偏好模型的预测目标是未来 7 天消费者在特定类目下是否发生点击，模型的输入除了常规的基础特征外，还采用了艾宾浩斯遗忘记忆曲线模拟消费者对类目偏好的衰减作为特征。基础特征是从消费者在 90 天的历史行为中提取不同时间窗口的行为，将特征划分为用户、类目、用户 & 类目三个大类别。每一类特征对行为进行统计分析，按照最近 7 天，30 天、60 天、90 天等时间窗口维度统计。特征体系如下所示：



将艾宾浩斯遗忘曲线理论应用到消费者兴趣类目中，消费者对类目感兴趣会通过平台上的一系列行为来体现，如点击类目下的商品、收藏 / 购买类目下的商品等。将用户对类目的行为考虑为用户的学习记忆过程，这个过程用户对类目的偏好程度同样也会随时间衰减。



图中竖轴表示学习中记住的知识数量，横轴表示时间（天数），曲线表示记忆量变化的规律。

衰减因子拟合方案：

1. 对用户某一行为（浏览 / 加购 / 收藏 / 购买），按时间 t 计算，在模型中采用的是离线数据按天统计。偏好 prefer 随时间 t 的衰减趋势如下图指数模型所示，即：

$$prefer(t)=\alpha_1+e^{((-t-\alpha_2)/\alpha_3)},t\geq 0$$

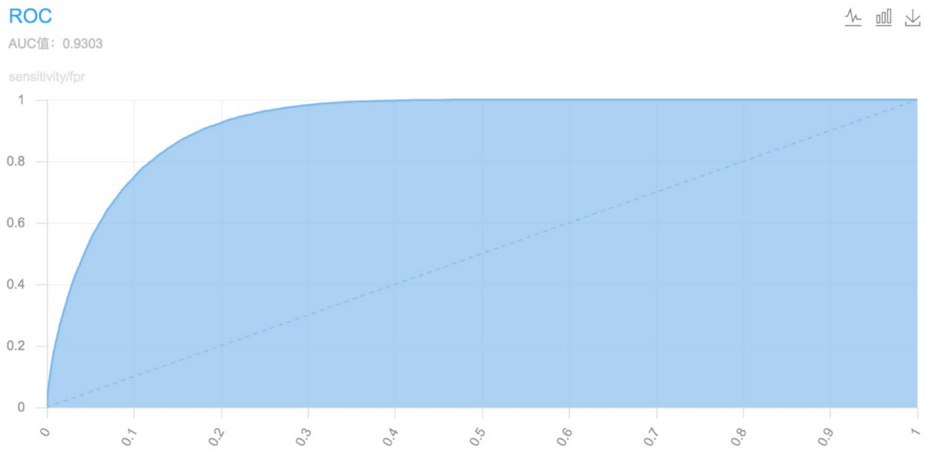
2. 对用户在不同时间窗口中的行为，统计行为频次。不同行为类型设定不同权重，从而影响衰减速率。例如，一共有 n 种行为，用户时间 t 内针对某一类目下的各行为发生的次数累计分别为 X_1、$X_2\cdots X_n$，则时间 t 内用户对类目行为量：

$$\chi_t=\tau_1*x_1+\ldots+\tau_n*x_n$$

3. 根据（1）、（2）计算用户对类目的偏好程度，时间窗口为 m 天：

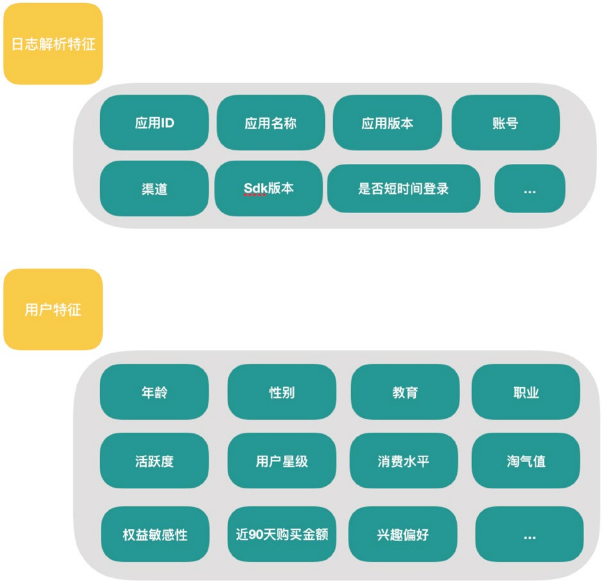
$$\psi(u,cate)=\sum_{t=1}^{m}\chi_t*prefer(t)$$

最终采用逻辑回归（LR）模型训练和预测，特征包含用户基础特征及拟合出的类目偏好衰减因子。模型效果如下图所示：

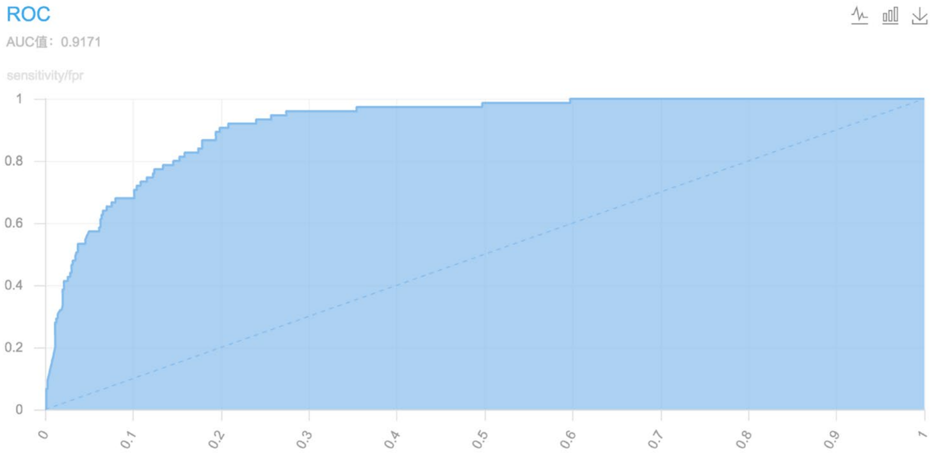


流失用户模型

流失用户模型基于用户在过去一段时间内在平台上行为数据的变化，预测用户流失可能性。该模型使用的主要特征包含应用平台基础特征、用户个人属性特征、用户的行为数据（例如用户在天猫、淘宝等平台上的浏览、加购、收藏、购买、品牌关注等行为）、不同时间窗口内的统计特征（例如移动端 3 天内点击数、7 天内点击数、15 天内点击数、30 天内点击数，pc 端 3 天内点击数 7 天内点击数等），具体如下：



流失用户预测模型采用随机森林（RF）模型进行训练，模型效果如下图所示：



展望

未来权益的发放应该会更加智能化，可以根据不同的营销目的，结合商家、品牌、行业、大促自身的特点（比如货品结构、消费者构成等因素），自动的选择权益类型的组合、进行权益设置、选择投放人群，在合适的时间、场景，将需要的权益以恰当的方式发放给消费者，同时也要考虑权益组合的复杂性，降低消费者认知、使用的门槛，提升购物的体验。

2017 双 11：开启智能全链路压测之路

长胜

导读：全链路压测诞生之后，整个阿里集团稳定性得到有力保障，随着集团业务蓬勃发展，新业务和应用不断涌现，为了能让井喷式业务得到稳定性保障的同时降低压测成本，达到“无人化”全链路压测，今年开始对智能压测进行探索。

智能压测概述

智能化压测，通过产品化、服务化、云化，一键完成阿里集团内外全链路压测准备和实施，保障集团内外全链路稳定；同时在常态化压测中，化身特种机器人，挑战系统承压能力，智能调整容量配比，快速定位问题。

如下图 1 所示，智能压测主要包含智能压测模型、自动化施压、预热系统化、压测云化、常态化智能压测五个模块。

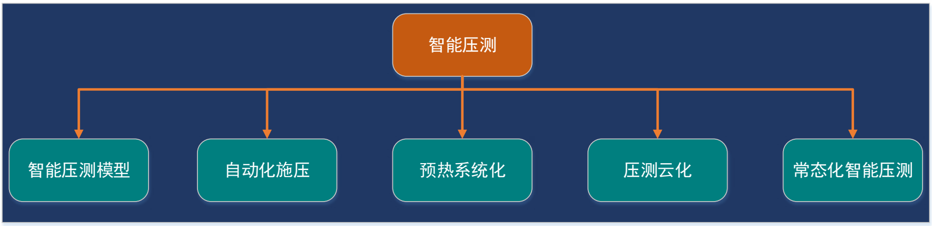


图 1 智能压测概述图

- 智能压测模型：高效提供一套准确的大促零点高峰压测模型
- 自动化施压：压测实施过程一键搞定，快速执行压测，准确发出目标量级的流量
- 预热系统化：确保各应用数据 & 系统预热全面且准确，保障全链路系统在大促峰值处于活跃态
- 压测云化：压测产品云化，为阿里集团内外各参与大促应用和系统提供压测服务
- 常态化智能压测：问题解决在平时，提高全链路压测效率

智能压测模型

全链路压测模型是全链路业务的抽象集合，直接模拟大促峰值模型，驱动整体压测。

智能压测模型，通过智能模块管理整条压测模型产业链，实现模型采集、预测、设计、构建整个过程的一键化智能操作，提升压测模型效率，同时在过程中将业务模型智能划分和计算，转换成可执行的压测模型，保障压测模型准确率。整体流程如下图 2 所示。

模型预测：采集往年大促业务数据，制定预测样本，通过预测算法，预测当前大促峰值模型。

压测模型智能划分和计算：将模型按照不同业务划分为可执行压测模块，统筹计算整体压测指标和各子模块压测业务指标（拆单比、主订单、子订单、购物车：立即购买、商品类型占比等），确保子模块和整体模型的一致性、准确性和完备性。

压测模型自动化调整：自动化调整压测模型，调整整体数据、业务指标等，达到一次压测验证多套模型的效果。

模型构建：一体化模型构建，根据模型参数，自动构建压测模型，生成压测流量，构建过程中可自动进行异常校验和模型数据校验，确保最终压测流量准确率。

压测方案：一键生成多单元多业务压测方案，压测方案可在多环境下执行，并且可根据业务要求，多模型组合执行。

智能管理模块：控制整体模型生产过程，一键操作，校验和修复异常流，控制模型构建对上下游系统影响，并使整体流程可视。

智能压测模型支撑大促效果：产出的全链路压测模型准确率达到 90% 以上，可在一天内完成大促压测方案设计计算和亿级别模型数据输出。

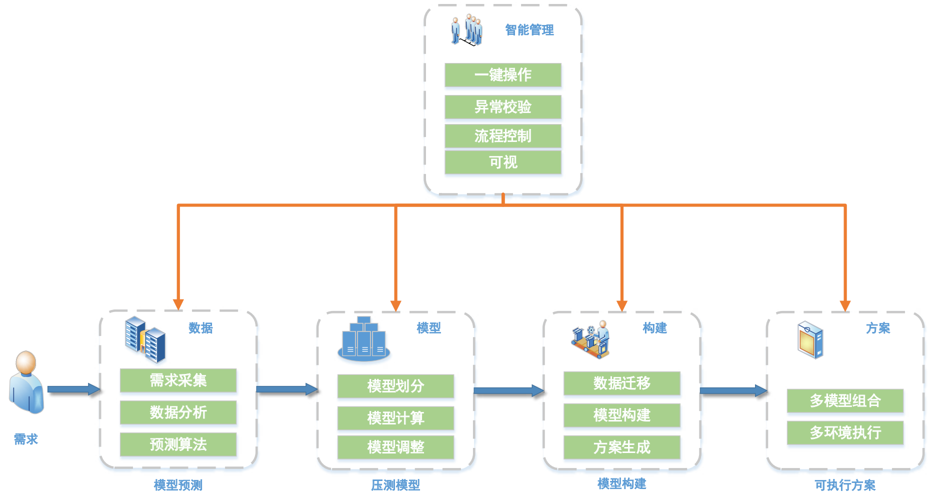


图 2　智能模型图

自动化施压

压测执行过程自动化，提高压测执行过程的准确率和效率，减少人为操作导致的异常和误差。

如下图 3 所示，今年在压测执行过程中，预热、预案、限流设置和动态调整、压测资源自动分配、施压整个过程均一键化操作，由系统校验执行结果及其准确性。通

过智能施压，今年的施压量级达到 1600 万 /s，预案执行准确率 100%，预热充分，限流准确。

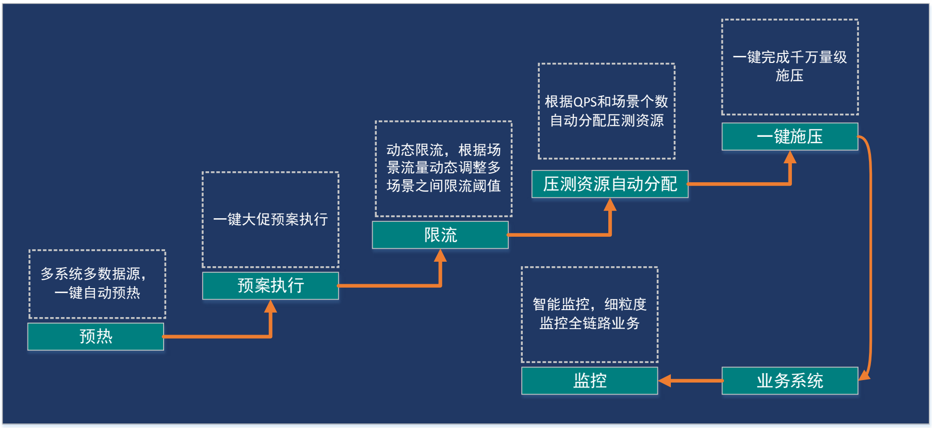


图 3　自动化施压流程图

预热系统化

大促期间预热必不可少，本地缓存和外部缓存均需要预热到位，避免大促峰值期间缓存被击穿，直接打到 DB 端产生雪崩效应，导致系统波动。

以往都是各个业务系统自行预热，中间可能会存在疏漏和预热不完善，今年推出智能预热系统，可覆盖核心应用及应用间关联的预热场景，一键实现全链路系统预热，事半功倍的同时也提升了预热的准确性。

通过智能预热系统，已实现亿级别数据的缓存预热、应用预热和 DB 预热，确保系统在 0 点峰值时处于活跃状态。

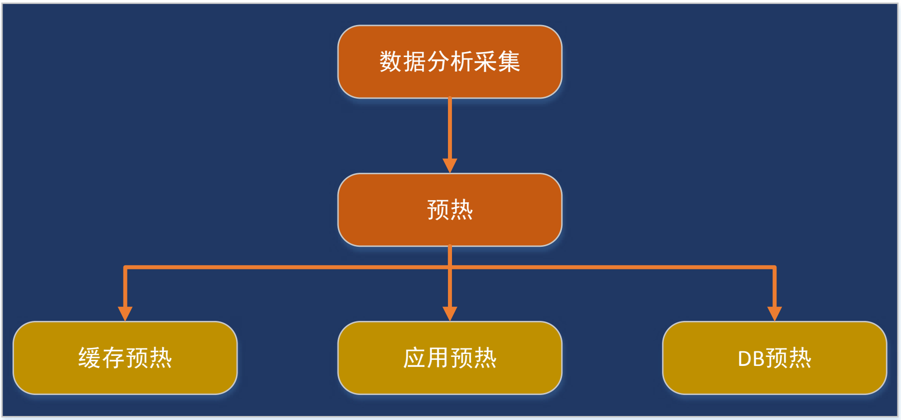


图 4　预热系统结构图

压测云化

压测云化，通过内部服务和上云实现外部服务，将集团内外的全链路应用全部纳入到压测范围中，即保障平台内部应用系统稳定，也保障关联的外部卖家应用性能稳定，在大促时达到整体稳定和双赢。

内部服务：产品化输出服务，为内部各业务提供可定制模型、容量弹性伸缩等个性化服务。

外部服务：将压测系统产品化上云，为外部商家提供压测服务，可实现数据、场景、模型、压测一键式操作，模型更接近大促峰值模型，压测环境和大促保持一致，压测效果更真实，保障卖家自身系统容量准确可靠。

以往的卖家自身系统压测过程中，是 mock 集团内部业务，直接模拟最后一步达到其自身系统的请求，同时使用的数据也比较单一，这中间会出现很多业务点覆盖不到，模型不完善，出现很多意想不到的问题；通过压测上云服务，可以为卖家提供真实的模型和丰富的数据，压测直接从最源头发起，把所有业务路径节点均覆盖其中，验证真实业务路径能力，确保各节点性能稳定，为卖家在大促峰值期间提供更可靠的业务能力输出，今年大促峰值期间各外部系统稳定，和集团内部应用交互顺畅，往年出现的问题彻底消除。

图 5　压测服务

常态化智能压测

常态化智能压测，在非大促态下全链路压测系统化身为智能压测机器人，对全链路系统进行固定频率的压测，沉淀全链路性能基线，及时发现系统瓶颈和定位原因，将业务应用瓶颈发现并消灭在平时，下图 6 展示了常态化智能压测流程。

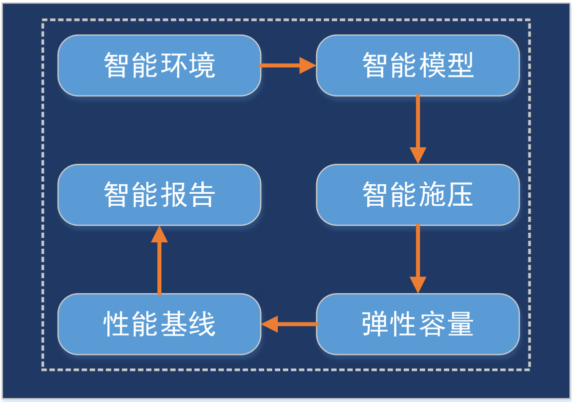


图 6　常态化智能压测流程图

智能环境：常态化压测在压测环境（也在生产环境中，有独立的一套和线上系统相同的配置）中进行，对线上流量无任何影响，操作时可一键将目标应用集群按比例隔离到压测环境中。

智能模型：压测模型采用大促模型，按照大促要求设计和执行压测策略。

智能施压：按照压测模型和策略和图 3 中的施压流程自动施压。

弹性容量：压测过程中，可根据当前系统表现进行弹性伸缩，确保在达到目标量级时，各系统按照预期性能指标调整到准确的容量。

性能基线 & 智能报告：在达到目标量级后，采集各系统性能指标和容量数据，沉淀性能基线，和以往基线进行对比，快速发现问题，并通过业务埋点监控定位问题可能出现原因，最终将压测数据、对比结果和问题原因自动录入报告并发送给业务方。

通过以上步骤，可实现无人值守的常态化压测，业务方在收到报告后，自行解决报告中的问题，下次压测再做验证，将全链路系统中的问题消灭在平时。

结语

在今年智能压测实施下，模型准确率达到 93% 以上，整体效率较去年提升 2 倍以上，为双 11 全链路压测节省 1000 人次工程师，同时保障了全链路系统稳定性，使今年双 11 峰值（32.5 万笔交易，25.6 万笔支付）如丝般顺滑。

今年是智能压测实施第一年，后面智能压测还会继续创新和使用更多智能化方法，让压测更高效更准确，让全链路压测“无人化”更近一步。

智能写手：智能文本生成在 2017 双 11 的应用

夜胧

一、引言

内容化已经成为淘宝近几年发展的重点，我们可以在手机淘宝 APP（以下简称手淘）上看到很多不同的内容形式和内容型导购产品，例如，“有好货”中的以单个商品为主体的富文本内容，“必买清单”中的清单，即围绕一个主题来组织文本和商品的长图文型内容，等等。不同于商品的形式，内容可以从不同的维度组织商品，从更丰富的角度去描述商品、定义商品，丰富了手淘的产品形式，提供给了用户更多有价值的购物信息。

随着手淘内容化战略的持续推进，我们也在内容自动化生成上持续探索，并构建了“智能写手”这个产品，旨在利用淘宝的海量数据，结合人工经验和知识输入，逐步实现内容的自动化、规模化生产，和人工编写的更高质量的内容一起，带给用户更丰富、更有价值的信息。经过一段时间的沉淀，目前智能写手在短文案生成、标题生成、商品推荐理由生成、图文型内容（清单）生成上都取得了一定的进展，期间针对若干文本生成的问题也进行了不同程度的优化。

在刚过去的 2017 年双十一中，智能写手主要做了两件事情，一是支持了大规模实时个性化生成双十一会场入口的短文案（下面称作“智能利益点”项目），保守估计生成了上亿的文案，提升了引导效率；二是进行了图文型清单的生产和投放试水，收集到了用户的直接数据反馈，验证了方案的有效性。

1.1 智能利益点

在每年的双 11 大促中，手淘首页、主会场等大促活动的主要流量通道上都会有很多的会场入口（参见图 1- 图 4），会场入口一般由三部分构成，分别是会场名称、利益点文案和商品图片素材。其中，利益点往往表达了一个商品或者一个大促会场最

核心的亮点，是商家、运营提升点击效果的一个抓手。传统生产利益点文案的方式，有以下特点：

1. 受限于数量和人力成本，一个商品或者会场的利益点一般不会超过三个，大多数情况只有一个利益点，这有时会导致利益点文案和商品不匹配的 case 发生，影响用户体验。
2. 用户对一个商品不同的卖点或者说不同的文案表述的关注度是不同的，例如有人关注性价比，有人关注品质等等，人工编辑的较少的利益点文案没办法提供多样的信息，不利于引导效率的提升。

因此，这次双十一，智能写手和首页推荐算法团队、大促平台算法团队一起合作了智能利益点项目，分别在手淘首页人群会场入口、猫客首页人群会场入口、猫客首页标签会场入口、双十一主会场行业会场入口、双十一主会场标签会场入口等多个场景上线了智能利益点。几个场景样式详见以下图片，其中用红色虚线框起来的使用了智能利益点的会场入口的实际效果：

（图 1 手淘首页）

（图 2 手淘主会场 – 行业会场）

（图 3 手淘主会场 – 标签会场） （图 4 猫客首页）

我们在双十一期间做了分桶测试，相比使用人工编辑利益点文案的分桶，智能利益点的分桶在多个场景都取得了用户点击率两位数左右的提升，这个提升是在各个场景自身优化效果的基础上的额外提升，还是比较可观的，这也说明了文案个性化生成确实给用户带来了更多的有价值的信息。

1.2 图文型清单生成

在手淘中，图文型清单是一种重要的商品组织形式，可以理解为有主题的商品集合富文本内容，主要由人工编辑而成，生产清单费时费力，尤其在大促期间，要短时间内生产大量的清单更是一个很大的挑战。这次双十一，智能写手也参与到这个工作中，结合在文本内容生成上的沉淀，生产了少量单品盘点类型的清单，具体样式如下：

（智能清单 1）

（智能清单 2）

为了验证智能写手生成的清单的效果，我们在双十一期间小流量上线，和人工编辑的清单进行了分桶测试。对比人工编辑的单品盘点清单，智能写手清单在平均商品点击转化率上的表现要更好。

下面我们将分别介绍智能写手在智能利益点和图文清单生成两部分的工作。

二、智能利益点

智能利益点解决的问题是，给定任意一个商品，挖掘这个商品各个潜在的卖点，并根据挖掘出来的用户偏好，从商品卖点集合中圈定用户最感兴趣、最可能点击的卖点，然后基于这些卖点实时生成一小段 6 个字以内的文案。利益点生成的解决方案主要分为这么几部分：

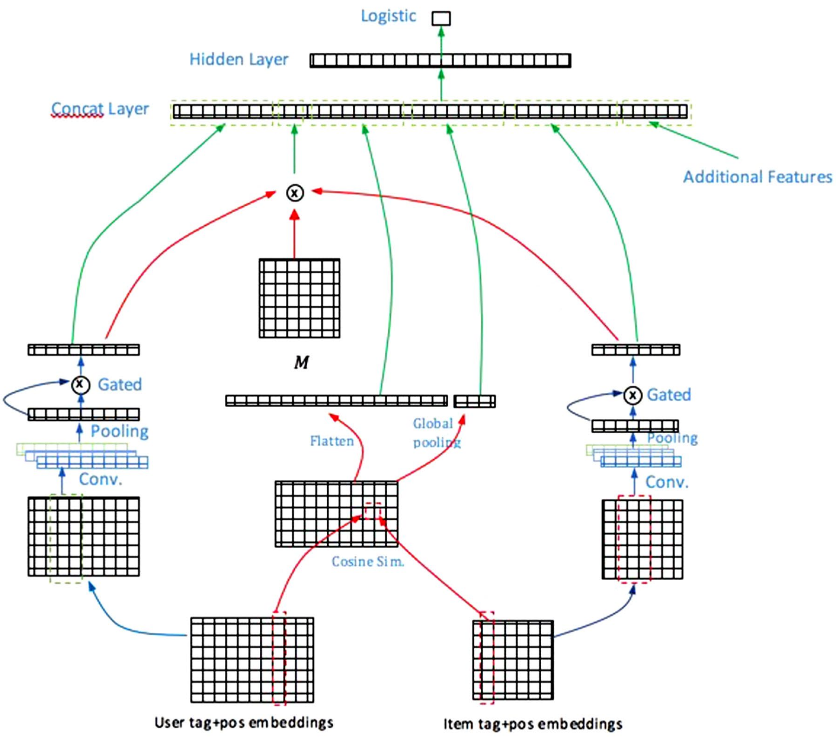
1. 用户的偏好挖掘：主要基于用户的离线和实时行为数据来做，通过挖掘得到用户的 TOP K 个偏好标签集合。由于线上系统性能的限制，我们不可能使用用户所有行为过的标签，于是我们构建了用户偏好标签的排序模型对标签进行优选。

2. 商品的卖点挖掘：卖点挖掘更多的依赖一些基础的数据，包括商品的标签库、属性库、人工编辑的信息等等。
3. 利益点文案的实时个性化生成：首先，我们提出了 PairXNN 模型，用于预估用户对一个商品的卖点的点击概率，然后，根据不同的场景要求选择合适的卖点，基于语义规则和人工设计的模板进行利益点文案的实时个性化生成。

下面主要介绍 PairXNN 的细节。

2.1 PairXNN 概要

在商品卖点的点击率预估问题中，我们把用户偏好标签和商品卖点都用文本的方式进行了表示，因此我们选择的 base 模型是 Aliaksei Severyn[1] 的工作，他们的工作主要解决短文本 pair 的排序问题。在经过不断迭代实验优化后，我们最终形成了我们的 PairXNN 网络结构，如下图所示：



模型包含几个主要的部分：

1. 用户偏好和商品卖点的语义表示：由于用户的偏好标签量比较大，如何对用户的大量偏好标签进行更深层次的偏好挖掘，是这个部分要解决的重点问题。
2. Multi-level 的相似度模块：在不同的语义层级上计算用户偏好和商品卖点的相似度。
3. Additional Features：引入人工定义的额外的特征，辅助模型效果。例如用户偏好的特征、卖点的统计类特征、用户偏好和卖点的 overlap 特征等。

整个 PairXNN 模型的训练和在线预测是基于我们内部自研的 XTensorflow 平台进行搭建。

2.2 语义表示

在对用户侧的偏好标签做语义抽取的时候，考虑到用户偏好标签的特殊性，它不是一个真正的有合理语义意义的句子，因此我们尝试了多种不同的语义表示的网络结构，包括全连接 DNN、和 [1] 一样的 CNN、Gated CNN[3]、self-attention[2] 和 tailored attention。

其中，Gated CNN 是对传统的 CNN 结构做了优化，加入了 gate 机制，可以决定哪些信息更为重要，需要保留或者说舍去哪些信息。而采用 Self-attention 则是考虑到对于用户的偏好标签序列，需要更关注全局的语义相关性。tailored attention 则是我们为了优化性能，简化语义表示网络所提出的新结构，因为智能利益点的场景都是重要场景，流量很大，对性能要求比较高。最终经过双十一期间的线上分桶测试，Gated CNN 在网络性能和效果上综合最优，于是双十一全量上线的模型中采用 Gated CNN 的语义表示网络结构。

2.3 Multi-level 相似度模块

除了上述对于 user 和 item 侧信息的映射和抽取，为了计算用户和利益点的相关性，我们从两个不同的语义层次对用户偏好标签和商品卖点的相似度计算，分别是：

1. 对用户偏好标签 embedding 层输出和商品卖点 embedding 层输出的 cosine similarity 计算。

假定用户侧所有词的 embedding 矩阵为 $U_{m\times d}$ ，商品侧词的 embedding 矩阵为 $T_{n\times d}$ ，那么两侧词之间一一对应的余弦相似度 (embedding 已归一化) 为：

$$S_{m\times n}(u,t) = U_{m\times d} * T_{n\times d}^{T}$$

我们还在这个基础上做了 global pooling，分别为 max pooling/min pooling/average pooling，得到 3 个数值。将上式得到的相似度打平后，与 pooling 层得到的结果 concat 成一维向量共同输入至下一层。

2. 对用户偏好标签的语义表示和商品卖点的语义表示计算 bilinear similarity。

定义一个矩阵 M 去连接用户侧向量 u ，商品侧向量 t ，公式如下：

$$sim(u,t) = u^{T}Mt$$

其中

$$M \in \mathbb{R}^{l\times n}.$$

为相似度矩阵。这相当于将 user 侧的输入映射为：

$$u = u^{T}M$$

由于此时 M 是可训练的，这样就可以更好的将 user 侧和 item 侧的空间靠近，提升相似度的准确性。

线上实验结果表明，两个层次的相似度叠加使用的 ctr 要优于单独使用。

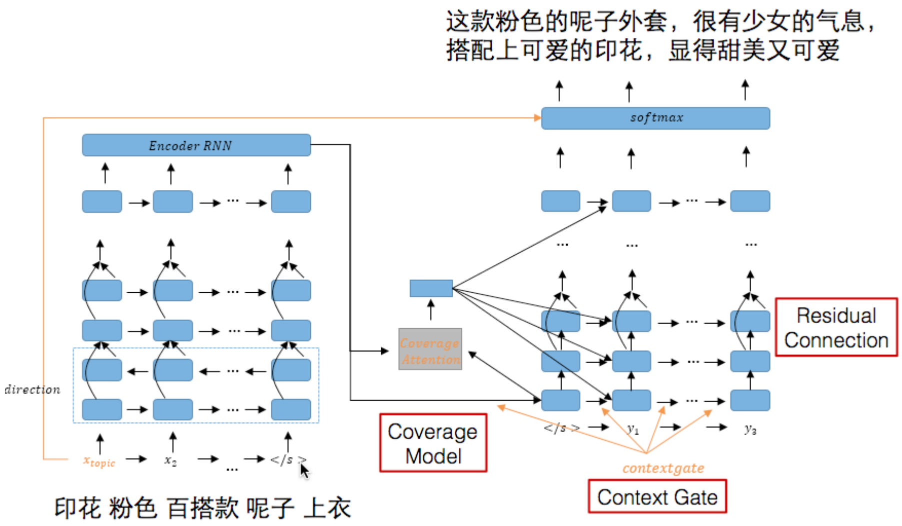
三、图文型清单生成

一个图文型清单一般都有一个明确的主题，围绕这个主题进行相应的文本描述和商品推荐。针对这个特征，我们的解决方案包括以下几个部分：

1. 主题。这个主题可以由外界（运营）输入，也可以基于我们主题发现所沉淀的主题库进行选择。
2. 选品。确定了主题之后，我们根据这个主题从精品库中选取和主题相关性高且质量不错的商品，然后以一定的目标组合成一个个的清单（一般一个清单包含 6-10 个商品）。
3. 商品推荐理由生成。为每个清单的商品生成一段 40-80 个字的推荐理由。
4. 标题生成。根据清单内的商品信息，给清单取一个概括主题又吸引用户点击的标题。清单标题要求相对简短，一般不长于 20 个字。例如："懒人沙发椅，沉溺初秋慵懒美时光"。

3.1 Deep Generation Network

图文型清单生成中的两个模块，商品推荐理由的生成和标题生成，我们把他们归类为自然语言生成（NLG）问题，都可以定义为依赖输入信息的文本生成问题。其中，商品推荐理由生成问题中，输入的是商品的信息，而清单标题中输入的是商品集合的信息。于是，我们采用了最近比较流行的 Encoder-Decoder 深度神经网络模型框架来解决，基于 Attention based Seq2Seq[5-6] 的 base model，最终形成了我们的 Deep Generation Network。



下面介绍几个比较主要的部分。

3.1.1　样本

样本的质量和数量是模型效果的基础，我们基于淘宝上的人工编写的商品推荐理由数据和清单标题数据进行了清洗，筛选得到符合我们目标的样本集数据。

3.1.2　coverage attention model[8]

在推荐理由生成中经常会出现多个内容重复描述同一个输入信息的情况，或者是对于输入信息在推荐理由中没有涉及。这个问题类似于机器翻译问题中“过译”和“漏译”的问题。在传统的统计机器翻译方法中，有 coverage set 的概念，去记录输入 source 文本哪些已经被翻译过了，而之后的模型主要考虑将没有翻译过的文本进行翻译。在深度学习中，是通过 coverage model 的方式和 attention model 做结合，达到这样的效果。

原来 attention 的计算方式如下：

$$\alpha_{i,j} = \frac{exp(e_{i,j})}{\sum_{k=1}^{J} exp(e_{i,k})}$$

$$e_{i,j} = v_a^\top \tanh(W_a t_{i-1} + U_a h_j)$$

其中 $a_{i,j}$ 是 attention 权重， t_i 是时间 i 的 decoder 的 state， h_j 是 encoder 的 state。

加入 coverage model 之后的 attention 权重计算如下：

$$e_{i,j} = v_a^\top \tanh(W_a t_{i-1} + U_a h_j + V_a C_{i-1,j})$$

其中相对于原始的 attention model，多了 coverage 的部分。这个部分如下计算：

$$C_{i,j} = C_{i-1,j} + \frac{1}{\Phi_j}\alpha_{i,j} = \frac{1}{\Phi_j}\sum_{k=1}^{i}\alpha_{k,j}$$

$$\Phi_j = N \cdot v(U_f h_j)$$

其中 Φ_j 是 Fertility 的概念，可以理解成一个 source 一般会被映射成多少个词的期望。

3.1.3 context gate[9]

在推荐理由的输出当中，模型的主体是基于 RNN 的 seq2seq 架构，那么在 decoder 的输出端的输出，主要受 2 部分影响：

1. 一部分是 encoder 的输入
2. 另一部分是当前 step 的前一个 step 的输出。

那么对于不同的输出，2 部分的影响应该是不同的，比如说，当前一个输入词是虚词时，主要的信息应该由 encoder 影响，但是如果前一个词和当前词明显有相关性时，当前词的主要应该由前一个词影响。所以，我们考虑加入 context gate, 对这种情况进行建模。

公式如下：

$$t_i = f((1 - z_i) \circ (We(y_{i-1}) + Ut_{i-1}) + z_i \circ Cs_i)$$

其中 s_i 是 source 的信息 embedding 之后的输出， t_{i-1} 是前一步的 decoding 的隐层 state， y_{i-1} 是前一步的输出词。$We(y_{i-1}) + Ut_{i-1}$ 是 decode 的前一步输入， Cs_i

是 source，利用 z_i 来决定下一个输出和那一部分关系比较大。

3.1.4 Beam Search

在前文中提到用 RNN 生成语句时，在每个时刻取输出结果中概率最大的词作为生成的词，是比较 greedy 的做法，没有考虑词的组合因素，因此，我们在 seq2seq 的实验中也尝试了 beam search。beam search 只在 predict 的时候使用，举个例子，当 beam search size=2 时，每个时刻都会保留当前概率最大的两个序列。beam search 在实践过程中很有用，它提供了一种很好的对生成序列进行干预的基础，一方面你可以对 beam search 的候选集的选择以及最终序列的选择做定制化的处理，比如你的选择目标，另一方面，对一些模型还不能完全保证解决的 bad case（例如重复词出现等），可以在 beam search 中进行处理。

3.1.5 CNN

对于我们生成清单标题的问题，由于输入是多个商品的文本内容，商品文本之间并没有真正的序列关系，反而更需要一个类似主题特征抽取的部分，从而能根据主题进行标题的生成。而 CNN 在句子分类已经有不错的应用[7]了，于是我们在清单标题生成问题中，采用了 CNN 作为 Encoder，实验结果也表明 CNN 比 LSTM 在标题生成的主题准确率上要高。

3.1.6 Reinforcement Learning

我们在训练和预测的时候会碰到下面 2 个问题：

1. 训练和预测的环境是不同的，训练在 decoder 的每次的输出，依赖的是前一个位置的 ground truth 的输入，而预测的时候是前一个位置 predict 的输出，原因是训练时候如果就依赖 predict 的结果的话，会造成损失累计，训练非常难收敛。
2. 我们的评价目标是 BLEU[11] 值，这是整个句子生成之后和样本之间的对比，而我们在训练的时候是对于每一个位置的 predict label 计算 loss，那么造成了评价和训练目标的差别，并且 BLEU 是一个整体目标，相当于是个延迟的 reward。

综上所述非常适合利用 reinforcement learning 的方式[10] 来解决。对于这样一个强化学习问题，首先我们定义这个问题的 3 个要素：

1. action：每一个 timestep 选择的候选词
2. state：每一个 timestep 的 hidden state
3. reward：最终的 BLEU 值

算法流程如下：

1. warm start：依旧利用原来的方法去训练模型，达到相对收敛的状态。
2. 逐渐在 decode 的末尾加入强化学习的方式，例如从倒数第一个位置加入强化学习，当收敛较好了，再从倒数第二个位置开始加入。Loss 定义如下：

$$
\begin{aligned}
L_\theta &= -\sum_{w_q^g,\ldots,w_T^g} p_\theta(w_1^g,\ldots,w_T^g)\, r(w_1^g,\ldots,w_T^g) \\
&= -\mathbb{E}_{[w_1^g,\ldots,w_T^g]\sim P_\theta}\, r(w_1^g,\ldots,w_T^g)
\end{aligned}
$$

3. 选择的 action 的时候，使用的是 KNN 的方式。本文是使用 REINFORCE 算法，是 policy gredient 的方式，并且文本的 action 空间非常大，所以比较难收敛。我们使用原来的 predict 方式打分，分数高的 N 个词作为候选词。然后这些词和 policy gredient 选出的词，做 KNN，距离是 embedding 后的距离，选择距离最近的作为 action。
4. 最终，除了第一个 timestep 还保留着期望的输入，其余都将是强化学习的方式。

3.2 效果展示

这里展示部分在测试集上生成的标题和推荐理由，给大家一些直观的感觉：

清单标题

- 卫衣，穿出你的青春活力
- 加绒牛仔裤，让你的冬天更有范
- 牛仔外套，穿出帅气的你
- 羊羔毛外套，温暖整个冬天
- 穿上格子装，让你秒变女神
- 职场新人，职场穿搭指南
- 穿上白衬衫，做个安静的女子
- 穿上蕾丝，做个性感的女子

商品推荐理由

这件针织款连衣裙采用了 v 领的设计，露出性感的锁骨，性感显优雅，衣身的撞色拼接，丰富了视觉效果，更显时尚感。

简约的圆领设计，修饰颈部线条，中长款的设计，显得优雅又大方，干净素雅，展现出清新的文艺风格，在端庄中流露出一股优雅的气质。

假两件的设计，让你的身材更加的修长，宽松的版型，穿着舒适，不挑身材，时尚百搭，轻松穿出时尚感。

四、展望

智能写手在双十一的智能利益点和图文清单生成上拿到了初步的效果，但是仍然还存在很多问题待解决，后续我们将在如下方面继续探索和优化：

1. 效果评估。现在采用 BLEU、覆盖率、准确率、人工评测结合的方法来评估效果，但 BLEU 和实际目标不完全一致，人工评测成本又较高，需要有更好的评价方案。
2. 更丰富的输入信息。引入包括商品图像、用户评价等在内的信息，除了可以解决输入输出的不一致外，还能给用户提供更有价值的内容。
3. 语言生成理解。通过模型的可视化，可以分析 bad case 的根源，优化模型。

4. 机器生成方面目前还有描述的准确度、多样性问题需要解决，另外考虑到很多缺少足够样本的业务也有生成的需求，模型是否能具备迁移能力也是一个可能的方向。

五、关于团队

阿里巴巴推荐算法团队目前主要负责阿里电商平台（包括淘宝、天猫、Lazada 等）的商品及 feeds 流推荐，其中用户导购场景个性化，首页首图个性化、猜你喜欢、购买链路等场景每天服务数亿用户，涉及智能文本生成、流量效率提升、用户体验、提高商家及达人参与淘宝的积极性，优化商业生态运行机制。

欢迎热爱算法，对业务有好奇心，有合作精神的同学一起工作、成长。简历可投邮箱：

shaoyao@taobao.com

或者 guli.lingl@taobao.com

或者 jinxin.hjx@alibaba-inc.com

翘首以盼，等你来信 ~

六、参考文献

[1] Severyn A, Moschitti A. Learning to rank short text pairs with convolutional deep neural networks[C]//Proceedings of the 38th International ACM SIGIR Conference on Research and Development in Information Retrieval. ACM, 2015: 373-382.

[2] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need. arXiv preprint arXiv:1706.03762,2017.

[3] Dauphin Y N, Fan A, Auli M, et al. Language modeling with gated convolutional networks[J]. arXiv preprint arXiv:1612.08083,2016.

[4] Luo W, Li Y, Urtasun R, et al. Understanding the effective receptive field in deep convolutional neural networks[C]//Advances in Neural Information Processing Systems. 2016: 4898-4906.MLA

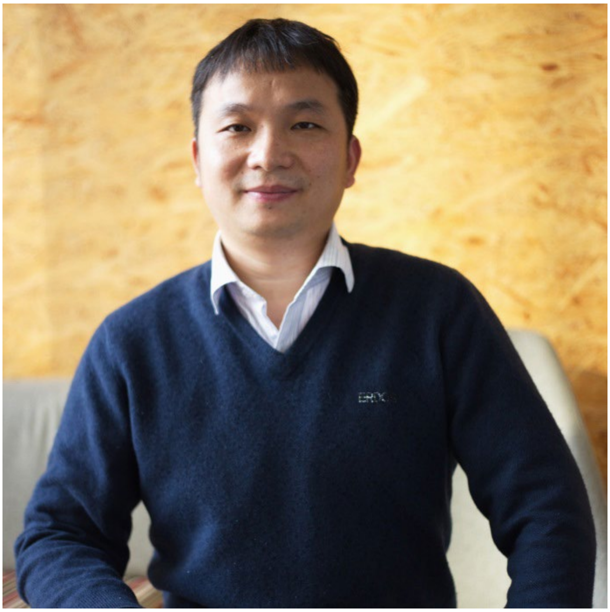
[5] Neural Machine Translation by Jointly Learning to Align and Translate

[6] Rush A M, Chopra S, Weston J. A neural attention model for abstractive sentence summarization[J]. arXiv preprint arXiv:1509.00685, 2015.

[7] Kim Y. Convolutional neural networks for sentence classification[J]. arXiv preprint arXiv:1408.5882, 2014.
[8] Tu Z, Lu Z, Liu Y, et al. Modeling coverage for neural machine translation[J]. arXiv preprint arXiv:1601.04811, 2016.
[9] Tu Z, Liu Y, Lu Z, et al. Context gates for neural machine translation[J]. arXiv preprint arXiv:1608.06043, 2016.
[10] Sequence Level Training with Recurrent Neural Networks, ICLR 2016.
[11] Kishore Papineni, Salim Roukos, Todd Ward, and Wei-Jing Zhu. Bleu: A method for automatic evaluation of machine translation.

浅谈分布式存储系统 Pangu2.0：它让双 11 运维变得智能起来

省身



阿里云资深技术专家，2012 年加入飞天 Pangu 团队，
主攻分布式存储方向，推动了 Pangu2.0 在双 11 期间的全面落地

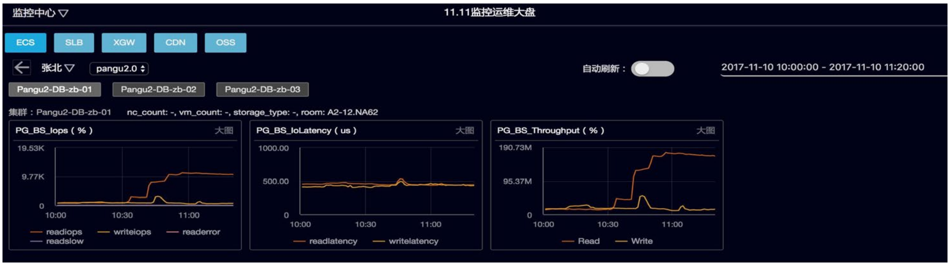
实测业务支持，在双十一中保持完善与稳定

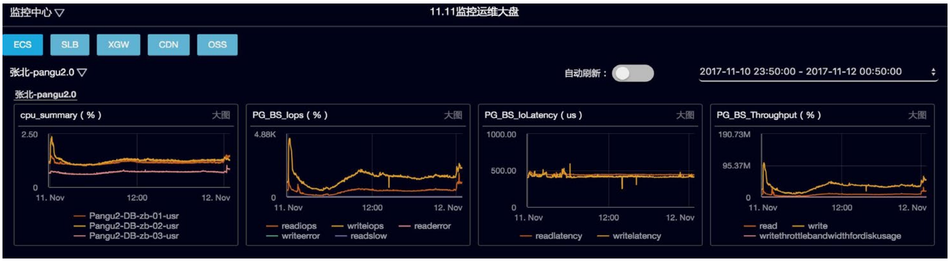
既然把双 11 作为一次对 Pangu 系统的战役，那么胜利的目标就是在业务支持方向达到最佳，事实上，Pangu2.0 在双 11 的业务支持主要由四个部分构成：

1. 集团 DB
2. 中间件
3. 列式数据库 Histore

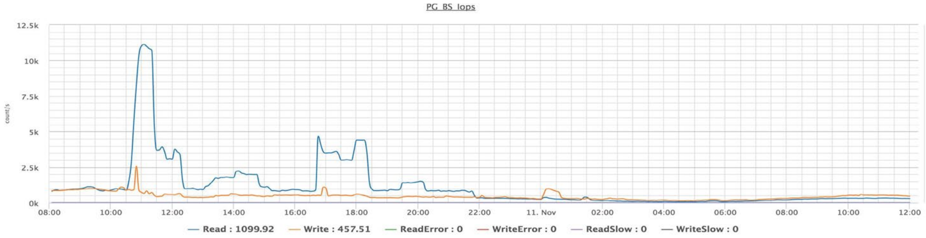
4. 对蚂蚁金服支付业务的支持

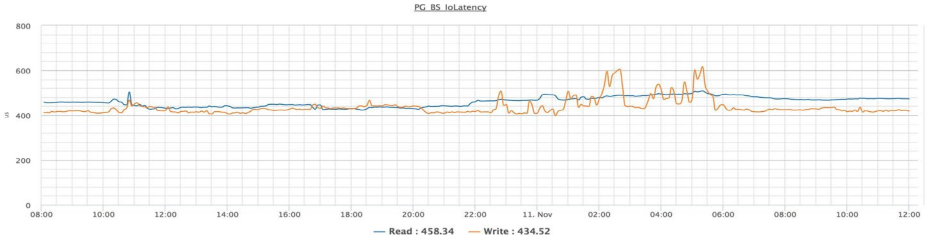
那么就首先从 DB 开始聊聊吧。根据下面的图片显示，双十一当天的数据库压力存在明显的峰值（即波峰与波谷），但同时可以看出，全天之内衡量 I/O 质量的「时延 Latency 值」却极为平稳的保持在 500 上下，没有明显的波动，如果用一句话来概括 I/O 表现的情况，想必就应该是「如丝般顺滑」，不存在惊喜，也不存在意外，且同样没有超出客户们的预期，监控人员对着平稳的波动表也抓不到什么特殊的数据——尽管这一天的 UPS 吞吐量波动堪称惊人，但时延指标却一直平稳。着充分的表明：系统的容量压力还远远没有达到上限，依旧可以对更大的 UPS 实现支持。



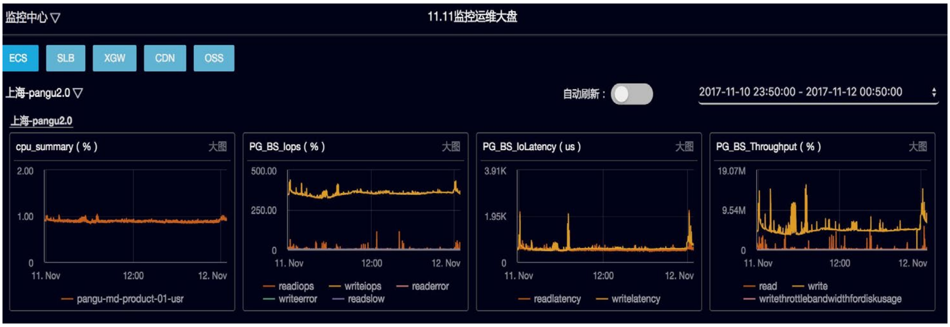


我们可以在 PG-BS 图上看到全天的 UPS 情况和时延情况。上图为将全部 UPS 平均到每台机器上的数据。很容易可以看出，UPS 的峰值出现在上午十一时前后，这是因为在这一时间点，工作人员对业务进行了一些热操作，导致此时的峰值一跃提升至平均值的十倍，但对应到下图中的时延指标，却只出现了不到十分之一水平的波动，全天读写表现出极为平稳的态势，仅仅到双十一当夜二时左右因为 I/O SIZE 的变化方才又一次带来大致十分之一的波动。





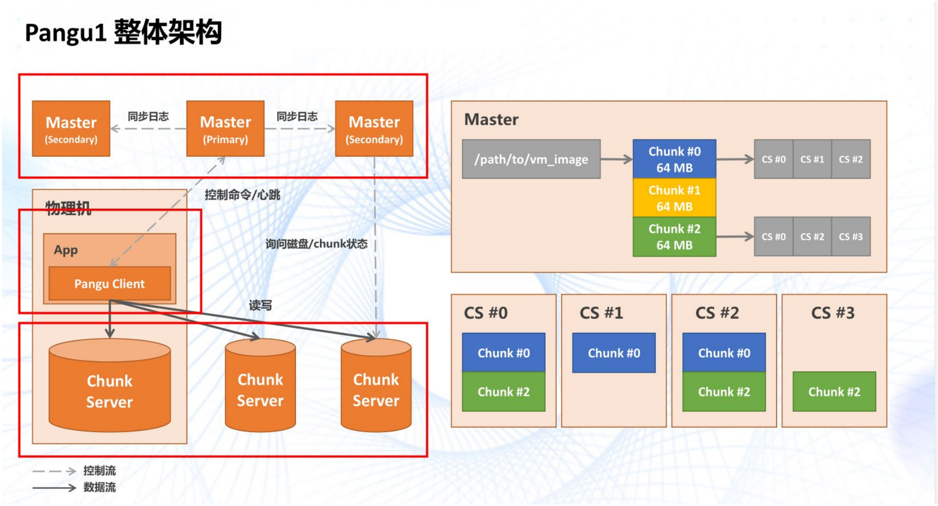
接下来谈谈中间件。起初因为集群负载偏高，无论是存储水位还是 UPS 水位都处于一个很高的水平，导致大家对此产生了一些担心，但实际值班时，我们对中间件时延的检测结果同样远小于预测，Latency 的抖动幅度只有用户预期的八分之一，曲线非常的漂亮。



Pangu2.0 诞生的原因，历史沿革以及相关架构

这里首先对 Pangu1.0 的整体架构进行介绍：它是一款经典的分布式存储架构，由几个部分构成，上层是 PanguMaster，下辖三台机器，负责解决存储原数据，命名空间以及具体数据的放置策略等问题，下面的部分是具体的存储节点，它的功能是

确定数据具体放在哪台机器上，并在这一层对数据进行存储，通常格式为 64 位。这是一个极为经典的架构，与业界的很多存储系统都很类似，例如 Google 的 GFS，Hadoop 的 HDFS 等等。他们的宏观架构都相差不多，具备着成熟的应用环境，完善的就近调度策略，其对线上业务的支持也已经持续了很长的时间。



推出 Pangu2.0 的原因

这是一个硬件和网络飞速发展的时代，最早做存储系统的时候，主流的网络制式还是千兆网。而如今，双 25GB 乃至 100GB 的网络已经逐步的投入使用。最早的存储媒介 HDD 机械硬盘需要 10 毫秒的时延才能成功进行访问，而现在 NVME 的盘时延则比之极低，十微秒之内就能完成一次写入，硬件的时延从毫秒压缩到微秒，使性能瓶颈的逐渐转移到传统的存储软件，传统软件无法适配新的硬件，令时延问题变得突出，必须进行革新来适配硬件的变化

可以举这样一个例子来方便说明——假设过去去美国一趟，飞机需要在空中飞行十个小时，中美海关各需要一小时的通关时间，旅程的总时长为 12 小时。但随着技术的进步，某款超音速客机在一小时就能直接抵达，那么整个旅程就变成了三小时，三分之二的通关时间就显得冗长起来。类比分布式存储系统：开始的时候，因为硬件的瓶颈，软件响应时间的长短并不是突出的矛盾，但随着硬件的提升，这一矛盾的重

要性也会日益凸显。我们能够得知：软件必须适应硬件的变化，这是创造良好用户体验的必要前提。

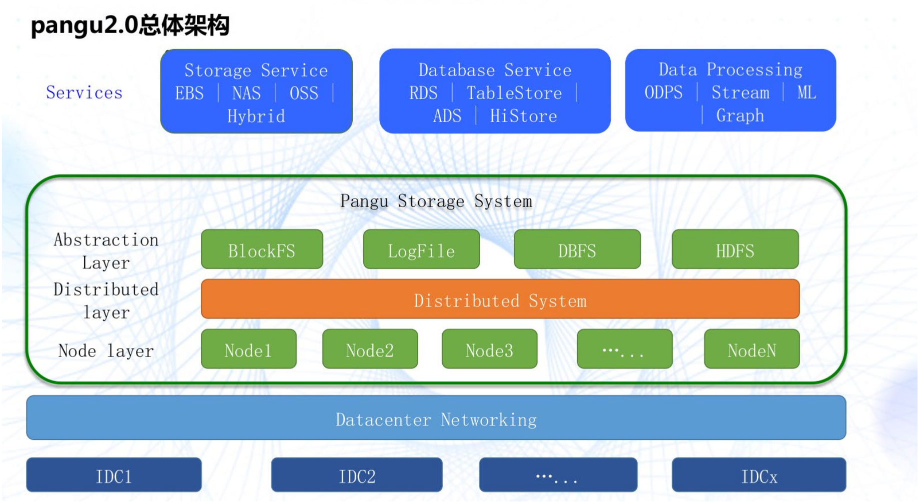
同时，近些年来，用户上传的数据一直在飞速的增长，分布式系统所覆盖的文数件也从十亿级跃升到了千亿级，单纯垂直方向的 Scale-up 的存储架构已经难以满足用户数据的需要，我们更多的开始需要一个能够水平扩展，不断满足千亿乃至更高级别需求，能够实现 Scale-out 模式的存储架构。

作为通用的存储平台，Pangu 系统一直在力求对更多的业务进行支持。完成多业务的的并行发展需要令模块分层更加单元化，以适应不同使用者定制化的需求。Pangu1.0 每次发布一个新版本都必须对每种不同业务需求进行综合考量，不仅时间上难以协调，且随着团队的规模逐渐扩大，这样一个单元和模块化不够细致的架构也愈发的不适于一个大团队的开发。亟需一个更加高效的架构，以更好的分层和更好的模块化来满足大团队快速迭代开发的需要。

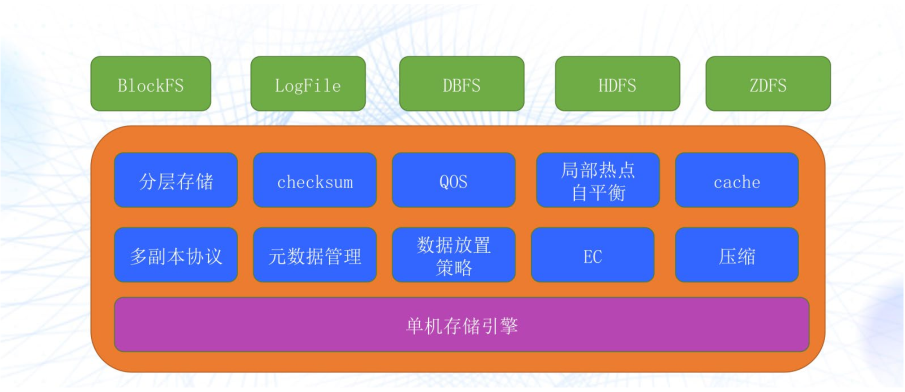
还有一点，随着近年来专有云，混合云的快速发展，对存储系统独立输出，轻量化输出的需求也越来越强烈，Pangu1.0 的输出不够轻量级，敏捷性也略显不足，这个过程也同样是需要加速处理的。

Pangu2.0 的总体业务架构

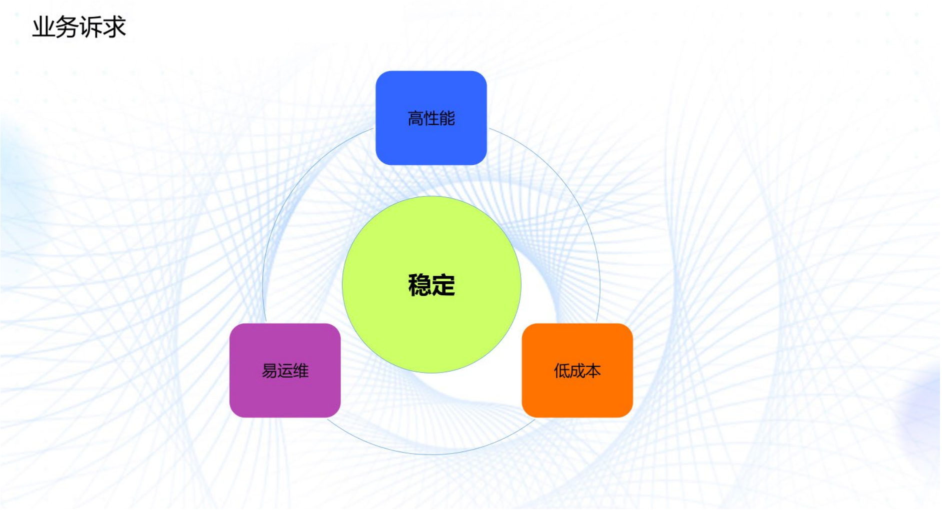
接下来，我们来聊聊 Pangu2.0 的总体业务架构。它的最底层是物理硬件架构与 Datacenter 网络，其上则是 Pangu 的存储系统，里面包括存储节点，分布式系统，存储系统内部的上层辐射出支持的多个业务方向，例如 Block FS，LogFil，DBFS 以及 HDFS ，整个系统的最上层则是目前主要的业务形式，包括存储服务、数据库服务、和大数据处理等一众分类。



详细分析 Pangu2.0 的模块划分，则可以将其分为两个部分，下层橙色的部门称为 PanguCore，即核心层，上层绿色部分则对应于各项业务的适配。PanguCore 的最底端是一个单机存储引擎，目的在于屏蔽左右硬件差异，保证为上层业务提供一致和接口相对稳定的内容，以此来解决基于硬件的高性能问题和新硬件的适配问题。Core 内部的蓝色部分下辖多个模块，包括多副本协议、元数据管理数据放置策略、EC、压缩、多种数据间的分层存储，以及自研的分布式 cache 等。两层架构的应用有效的克服了 1.0 版本的不足，使各个模块能够独立发布，提高了敏捷性。模块改革的耦合度低，团队战线布的非常宽，也更适合一个大团队进行共同项目的开发。



解决核心诉求 做到用户满意



存储系统的核心诉求无外乎几点，重中之重的稳定性、性能尽可能高、成本尽可能低，运维难度同样越低越好。在接下来的文段中，我们将针对这些用户永恒的追求，来详细的介绍 Pangu2.0 在这四个部分做到的一些成绩。

稳定性：高可用

首先是在稳定性方面的成果。面向线上众多的块存储集群，我们要在日常工作中频繁的对其进行扩容、缩容、下线、机器维修、集群整体迁移、软件版本发布和变更等各式各样的操作。在自始至终的整个过程中，我们实现了全年零故障，完全保障了业务的稳定性格。

现在正在进行的“系统盘云化”工作也是一项良好的佐证，未来，我们的服务器物理机将不再采购系统盘，而是直接通过协议导出做成云盘，这同样充分体现了我们对稳定性的掌控，一旦突发问题产生，我们的终极目标就是让用户需要意识不到存在稳定性波动的可能。实现这点需要内部极为复杂的技术手段和管理手段，以及反复进行的尝试。

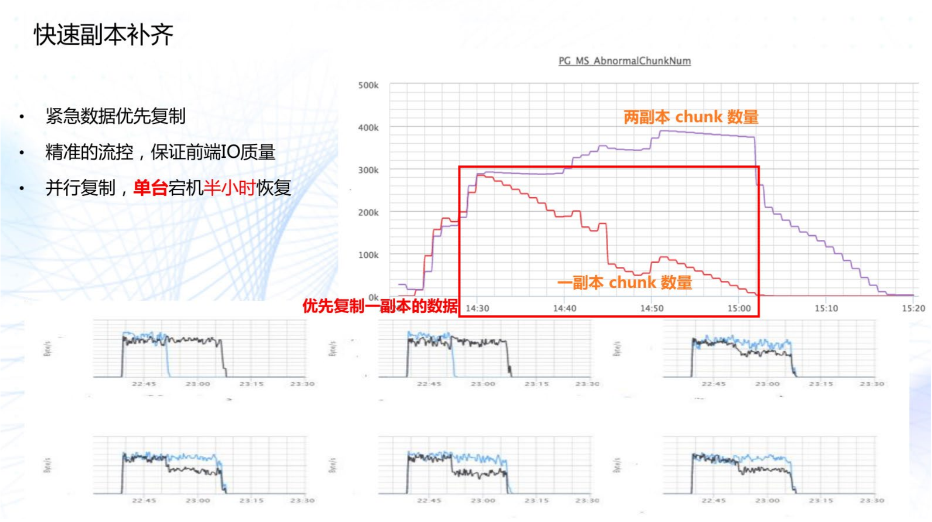
我们的另一个成果在于使基础设施完全消除了故障依赖，没个数据三副本都分散在三个 rack 上，每个 rack 都是独立的供电和网络单元，发生供电或交换机故障时不

会影响全部数据，对用户读写也不会产生影响。以及保持健康状态，即对所有硬件进行自动化管理，在用户感知前就能够自动发现问题和解决问题，对故障硬件自行触发汰换流程，实现无人为的有机循环。

说道稳定性，实现单机的一致性就是保持数据稳定不可或缺的一部分：日常使用中，数据和日志同步落盘写入一个存储，必须保证二者一致来保证读写稳定，即数据写透盘片后，必须进行掉电测试。

第一是进行端到端的数据校验，消除静默错误。每次数据写入都要通过一个 CRC 来进行保证，不管硬盘，内存还是 CPU 网络出现错误，用户在读取数据的时候要能够知道数据是错的，绝对不能将错误的数据传递给用户。

第二是快速副本补齐。在某些紧急情况下，我们需要进行对于三副本的数据复制，集群交换机故障或者掉电的出现都属于这一范畴。这一过程非常精细，且具备严格的优先级区分。发生硬件故障后必须先复制高优先级的（例如三个副本只余其一）。在大范围掉电条件下，先进行单副本 chunk 数的降低，随后才进行单双副本的共同复制。该过程中存在精准流控，能够反复权衡流量的使用，保证复制的同时前端用户的 I/O 依旧维持在可用度很高的状态，并采取并行复制的方法在半小时内完整恢复单台宕机的全部数据，从而尽可能的淡化影响。

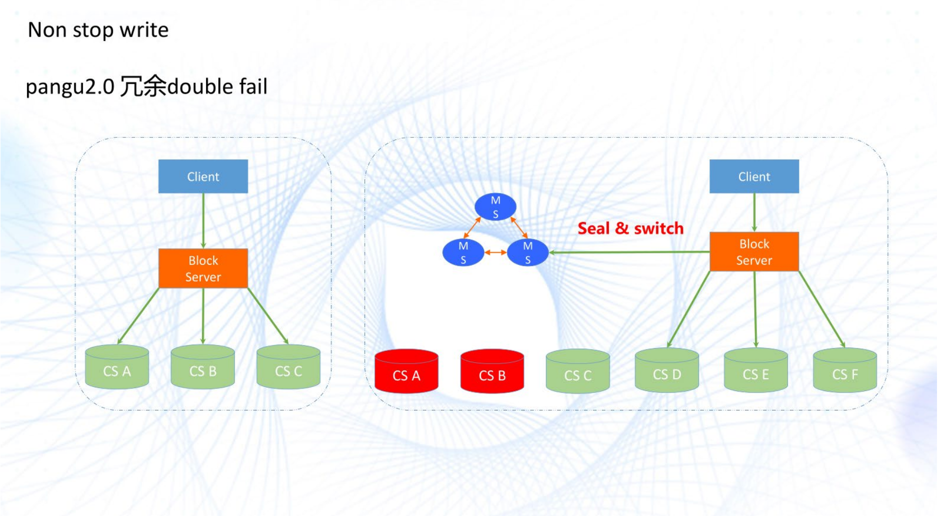


前文中，我们讲了用于维持稳定性的一些大体技术手段，而面对系统抗压能力的测试，我们也同样会采用非常严格的手段。从图中可以看到，平均每台机器都在每秒上百个 UPS 的条件下各种自杀进程（包括但不限于 cs、bs、同时杀两台 bm 等）的 failover 测试才敢于交付给用户。这一套测试和管控成熟，无论下面的进程如何 failover，上面的 UPS 始终处于一条直线上，波动极小。

除了自杀进程，rack 掉电的模拟往往会显得更加的极端，每个版本发布前我们都要进行 rack 掉电的模拟：直接关掉涵盖 48 台机器的 rack 集群，并测试其恢复的过程，实际结果表明，掉电的机器能安全的将负载转移到其它机器上，待掉电的 rack 恢复后再将负载转移回来，全部掉电机器的功能都能在一分钟之内恢复。整体过程对用户使用上的冲击很小。

还有另外有趣的一点，这比较像一道概率题：通常情况下，在一个集群的规模内，非常小的时间窗口内（例如一台机器重启的时间内）两台机器 failover 的概率应该是可以忽略不计的，但随着样本的容量增加，小概率事件长期积累就会必然发生，很短的时间窗口内两台机器同时 failover 的糟糕境况也会时有出现。如果一个客户端同时写入 A、B、C 这 3 台机器，但 A 和 B 都出现了 failover，就会只剩下 C 的一份形成 I/O HANG。解除的唯一方法就是进行数据复制，将 C 中的数据复制到 D，

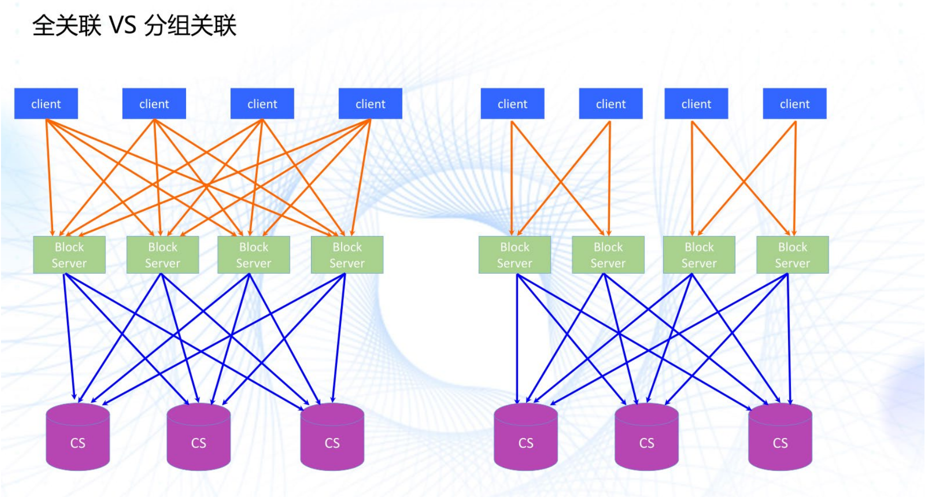
形成至少两份数据以确保其安全，但是在复制的几秒钟的时间内，这一 I/O HANG 无法解除，可能会严重的影响用户业务。这一问题在 Pangu2.0 中得到了妥善的解决：我们直接假定 double fail 常在，默认用 Block Sever 对 A、B、C 进行写入，如果 A 和 B 同时出现错误，就直接切换文件，把数据写到一个其它流上（例 D、E、F），从而将用户干扰下降到毫秒级别。



我们知道，在工程领域，黑天鹅事件的发生通常是无法避免的，永远会有超出认知范围之外的意外在前方等着我们。这种事情发生时，我们要考虑的内容就会变成怎样在既定条件下将损失控制在一个最有限的范围内。对此，我们针对 Pangu 系统的 1.0 版本进行了优化，将元数据管理划分成两个部分：即 Namespace 和 Meta Server。把原来三节点的元数据管理分成多组，离散的分担到所有存储节点上去，即使其中的一组发生完整故障，也只会影响一小部分用户，可以根据内部的其它策略进行及时的迁移。确保无论任何一个组件发生故障，整个系统依旧能维持在可用的状态，并做到在最短时间内恢复，怎样减少爆炸半径，在极端事件发生的情况下保证系统柔性可用。

接下来将问题细化到具体单个存储节点的 failover。我们此前的调度是全局调度，它存在一定的缺陷：如果一台机器出现宕机，那么这台机器上承载的全部 I/O 流都会受到影响，甚至会在极端情况影响所有的用户。而如今，我们进行了一个分组关

联，将部分用户和某个存储节点进行链接，成功使机器错误的影响范围缩小至最低，如果集群规模较大，影响用户的比例就会变得极低。



技术手段的发挥离不开相关标准的制定，硬件和环境上的标准也是稳定性足以实现的重要部分。线上集群诸多，由于历史原因的影响，各类硬件，网络，内核，和应用配置等信息的跨度都很发散，造成了隐形的危险：任何一个变化的影响都很难百分百的提前进行覆盖和评估。对此，我们着力推行环境标准化，标准服务化，一切内容都只有先遵循标准才能进行上线，从而真正把环境标准落到实处。

此外，改进稳定性的手段还有不少，比如，我们会组织两个工程师团队形成攻守同盟，抛开经验，发挥想象，蓝军的任务是在系统不同单元制造 failover，例如磁盘，网络，内核等，红军进行防御，并在接下来评估对错误系统和用户的影响。通过这一内部竞争显著提升了系统的稳定性。

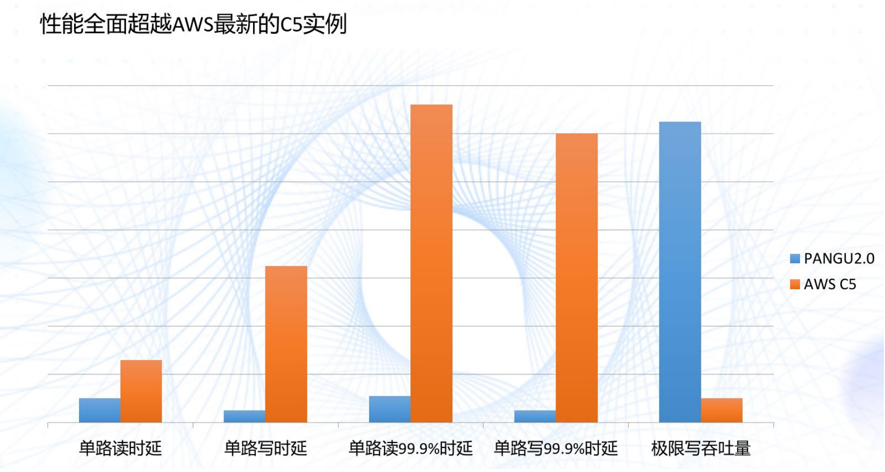
针对运维操作的响应性，我们制定了一个「双十」标准作为最后的两道防线：任何时候收到告警，团队都要在十分钟内响应。且一周收到的告警数不得超出 10 条。在技术人员的长期坚持和推行下，这两个标准都取得了成功。

高性能：对竞品和自我的同时超越

先看一组客户对于 Pangu2.0 性能的反馈。

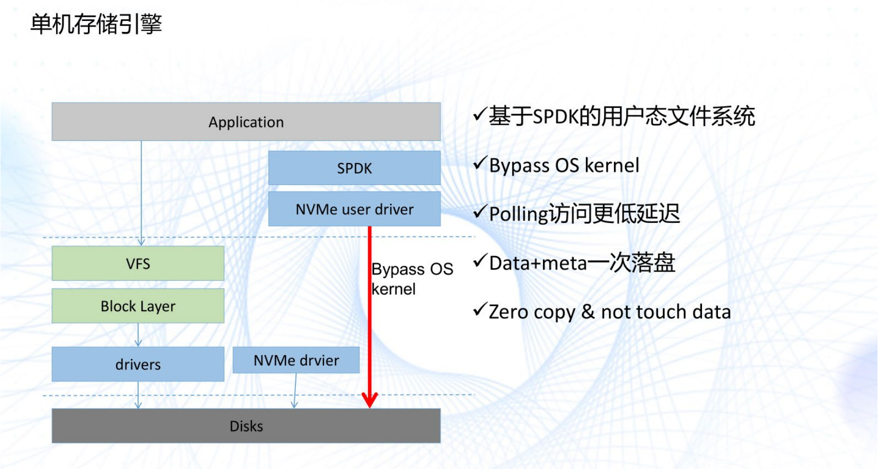
1. DB: XDB+Pangu2.0 取得了超 Aurora 的 4 倍以上的 TPS。后续会和 Pangu2.0 在性能，成本，高可靠等领域深度合作，为用户提供更好的 DB。
2. 中间件：镜像加速项目每次镜像 push、pull 时间从 50 多秒缩短至 1 秒内，一个月全集团走 aone 的发布可以节省数百人天。Pangu 2.0 性能和稳定性全面超越竞品，今年合作磨合非常顺利，明年大规模的存储计算分离，「离在线」和「在离线」混部会大力推进。
3. 分析型数据库：pangu2.0 非常靠谱，相比同规格的物理盘，为分析型数据库带来至少 10% 的性能提升，后面分析型数据库的存储会全部迁移到 pangu2.0 上。
4. ECS 云盘产品：性能大幅超越 AWS 最新的 C5！存储领域提前实现对 AWS 的技术超越。

下面是对 Pangu2.0 和 AWSC5 实际性能的表格对比，可以很直观的看出，无论是单路读时延、单路写时延；还是单路读 99.9% 时延、单路写 99.9% 时延，蓝色的 Pangu2.0 都要明显优于橙色的 AWSC5，极限吞吐量更是超越了一个数量级。



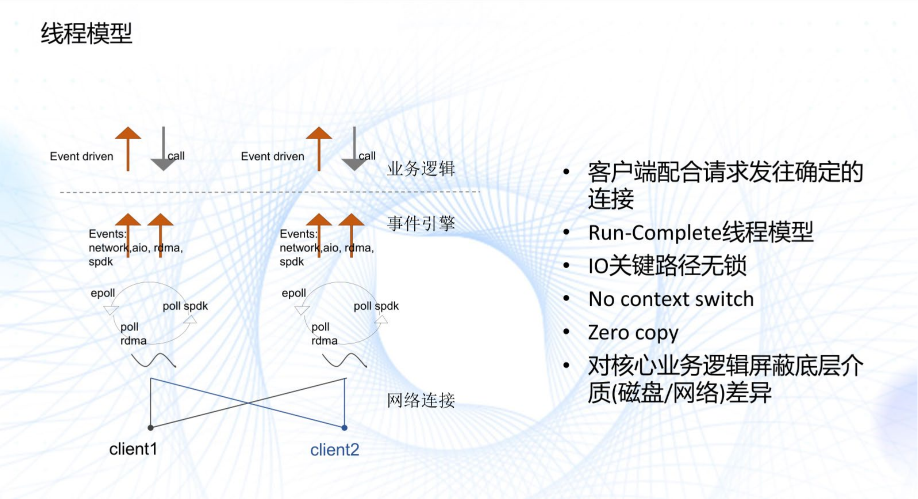
这么好的性能数据从哪儿来

首先，Pangu2.0 拥有自己的单机存储引擎 Bypass OS kernel，它是一个基于 SPDK 的用户态文件系统，区别于使用 VFS、Block Layer 和 drivers 进行传递的传统文件系统，Bypass OS kernel 直接将文件返回 NVME 盘，使用 Polling 方式进行访问来降低延迟，Data + meta 直接一次落盘，整个过程中无需进行任何拷贝。



网络上，Pangu2.0 不再使用基于内核的 TCP，而是利用 RMDA 网络 Bypass 掉内核，省略系统调用的过程。同样使用 Polling 方式进行访问，全过程零拷贝。

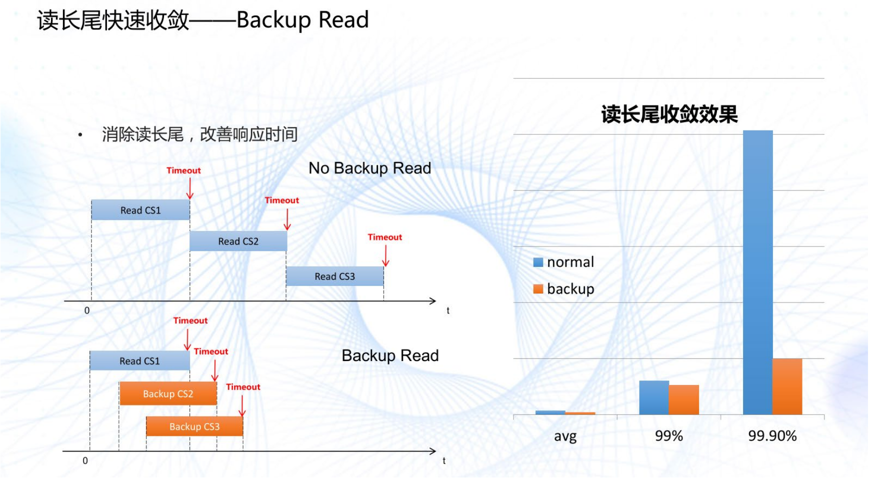
另外一件很有趣的事情就是在线程模型上的优化，我们将客户端和服务端进行了一些配合，客户端的链接由指定线程处理，形成 Run-Complete 的线程模型，从 I/O 请求到业务完成全部在一个线程内转完，没有任何上下文切换，且时间可控、关键路径无锁、无数据拷贝。



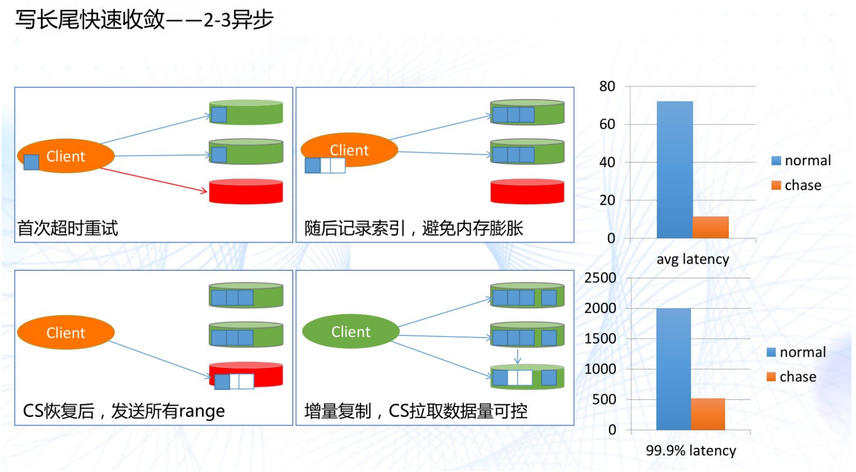
我们还真正实现了 I/OPS 与云盘空间的解耦，现有的云盘最大 IOPS 值为 20000，此前，如果用户需要使用 2 万 I/OPS，则至少需要购买 600GB 空间才能实现。而 Pangu2.0 彻底实现了 I/OPS 与空间的解耦，只需 128GB 的云盘即可实现超百万 I/OPS，对 I/OPS 需求大，空间需求小的用户尤为适用，避免了维度浪费。只要愿意，多大的盘都能得到超高的 I/OPS。

前面的文段中，我们综合介绍了平均水平上 Pangu2.0 在高性能的方面举措，但评价 I/O 水平的指标除了平均水平外还有长尾收敛，我们往往希望长尾收敛的越快越好。同样，Pangu2.0 在长尾指标上也做了不少的建设。

第一是读长尾快速收敛，我们做了一个 Backup Read 的算法来进行优化，载入 Read CS1 后，如果短时间内没有返回值，那么会在极短的时间内直接载入 CS2，CS2 无返回值则继续读 CS3，只要有一个请求得到回复，我们就认为是响应成功的，就能在即使最坏的结果下也能把用户的读操作收敛到四倍的平均时间内，如图可以看出，在百分率达到 99.9 之后，读长尾的收敛效果极佳。

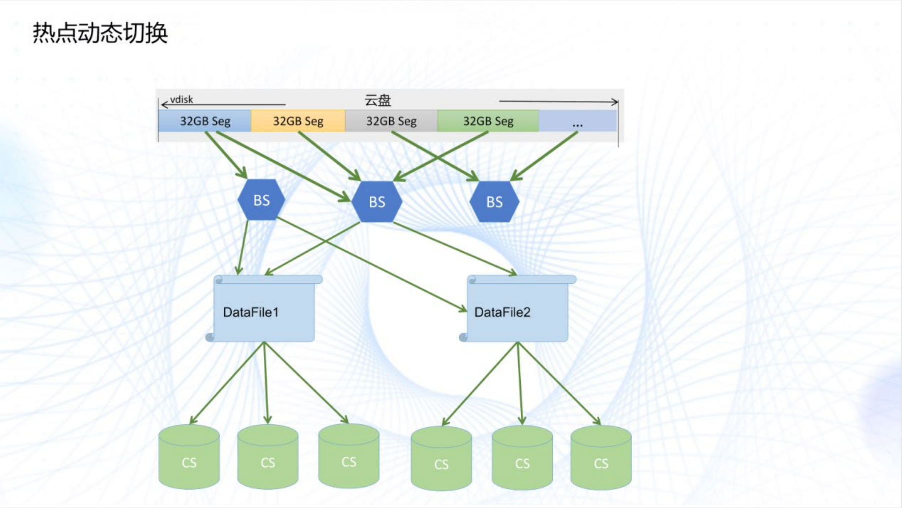


第二是写长尾快速收敛——这里采用 2-3 异步的模式进行。对于一个需要写入三份的文件，通常情况下认定写入两份即为写入成功，第三份可通过异步化操作的方式进行补充。假设文件写在 A、B、C 三台机器上，且其中一台真的发生了故障，那么写入 A 和 B 两份成功后，首先将信息返回用户，再对另外一份在客户端中进行 range 的记录（描述第三份有那些数据没有计入），再从后台向空置的 C 端进行推送，在这一条件下，即使系统中出现单机的抖动或故障，绝大多数用户也能够可以被系统过滤，从而可圈可点的降低时延指标。



此外，分布式系统中还有一个非常复杂并难以处理的问题：局部热点。一般情况下，规模较大的分布式系统都有多个租户同时使用，一部分节点会因为外界巨大的访问量而形成热点。例如图中的三台机器，如果有一台变成热点，前文中的快速读写可以让用户屏蔽掉这个问题，但如果情况再进一步，有两个节点都变成热点，读取数据可能影响不大，但写入的过程则会出现困难

这时，我们会引入一项多流技术。将因为形成热点而速度下降的流废弃，立刻切换到另外一个流里去。因为新出现的 Datafile 客观全新，所以就不会存在这个问题，这一切换过程只需要一个 RPC 的时间，可以做到用户基本无感知，如果问题出现在 BS 上，即 BS 所承载的 I/O 过量的话，就会在用户中产生一个较高的时延。这时我们同样会对 BS 进行切换，对用户的 I/O 的影响依旧可以控制在很小的范围内。



实际上，在很多维度中，基于云的 Pangu2.0 已经对物理盘实现了超越，例如：

1. 多流并行映射技术，使得云盘的吞吐量可以水平扩展，吞吐量大幅超越物理盘。只受限于客户端所在的网络带宽。
2. 空间上的水平扩展没有限制，云盘可以做到 PB 级别的容量，与最新的物理盘相比，有几个数量级的优势。

3. 能够通过一系列技术手段来优化长尾，做到长尾优于物理盘，物理盘的热点无法通过切走来进行优化，云盘让这一点变得可能。

低成本：稳定高效之外，经济依旧是特长

除了出色的稳定和优秀的性能之外，更低的成本也是 Pangu2.0 的一大特色，例如：

1. 全面支持 EC，从而能够把经典的 8+3 从三份变成 1.375 份。
2. 支持自适应压缩 根据特定算法筛选出能够压缩的文件，对其进行压缩。
3. 数据冷热分离，冷数据存储到廉价介质，热数据存到高性能介质，形成资源的合理规划。
4. 管控水平拆分到所有的存储节点，不需要独立的管控机型。
5. 云盘跨集群，将单集群的空间利用率做到极致，充分发挥云的优势。

Pangu2.0 的软硬件一体化工作也一直在同步进行，我们与 AIS 合作，共同研发了 USSOS - User Space Storage Operating System，并实现了很多之前所期待的目标：

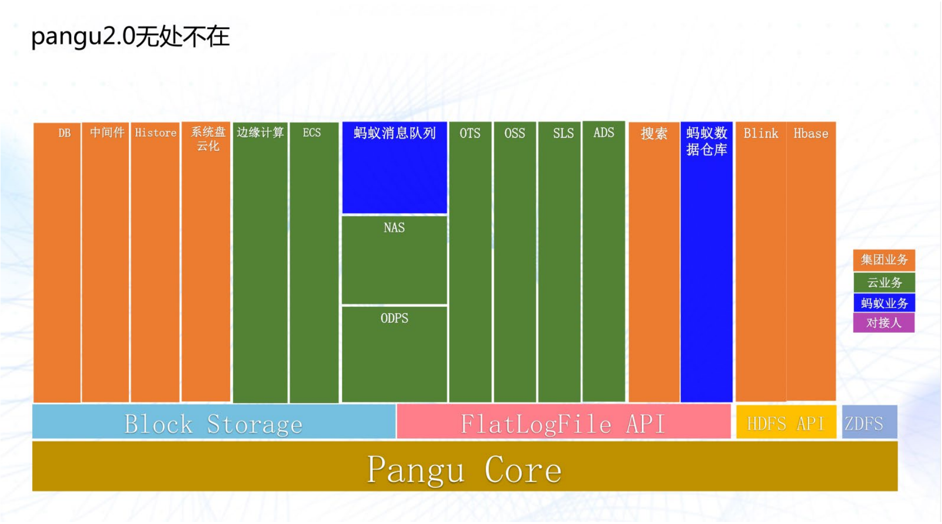
1. 建立一个统一的用户态存储软件基础平台
2. 实现对新硬件的快速适配，任何硬件只要在 USSOS 层进行适配就能直接应用，且对从前的软件和服务毫无影响，完全公开和透明。
3. 提升 I/O 处理效率，发挥 I/O 极致性能
4. 识别和冗余硬件故障，简化系统运维与管理，增强系统稳定性
5. 实现存储硬件的自主可控，降低供应链风险，降低成本，使用户能够选择低成本的硬件，同时更激进的使用新的技术。

易运维：将使用的压力同样降到最低

1. 高度的可运维性也是 Pangu2.0 不得不提的一个点，并在相当多的方面能够得到体现：

2. 所有存储节点故障自愈，无需人工干预 提前检测，自行报修，自动下线和复制技术 维修后自动重新上线
3. 管控故障自动迁移替换 管控节点自动替换
4. 运维高度自动化，ECS 线上人均可运维数百个集群，数万台服务器
5. 在支持集团业务中，我们承担所有存储的运维工作，把麻烦留给自己，把便捷送给客户。

文章的尾声，就让我们再次来回顾一下 Pangu2.0 在阿里云内部所有支持的业务，这是一个稳定而无处不在的平台，它将阿里的集团业务、云业务，蚂蚁业务和对接人串联在一起，我们完全可以这样进行描述——名为 Pangu2.0 分布式存储系统，切切实实的让双 11 运维变得智能了起来。



新基础

双 11 稳定性负责人叔同讲述：九年双 11 的云化架构演进和升级

叔同

阿里妹导读：阿里巴巴 9 年双 11 经历下来，交易额增长了 280 倍、交易峰值增长 800 多倍、系统数呈现爆发式增长。系统在支撑双 11 过程中的复杂度和支撑难度以指数级形式上升。双 11 峰值的本质是用有限的成本最大化提升用户体验和集群吞吐能力，用合理的代价解决峰值。面对增长如何发挥规模效应，持续降低单笔交易成本，提升峰值吞吐能力，为用户提供丝般顺滑的浏览和购物体验，这是极大的挑战。

今天，我们邀请了阿里巴巴资深技术专家叔同，分享九年双 11 的云化架构演进和升级。

叔同（丁宇），阿里巴巴资深技术专家，8 次参与双 11 作战，阿里高可用架构、双 11 稳定性负责人，阿里容器、调度、集群管理、运维技术负责人。

叔同：大家好，我是叔同，很高兴与大家分享阿里双 11 的技术发展。今天我们先来关注一个问题：双 11 推动了阿里技术的进步，它有哪些挑战?

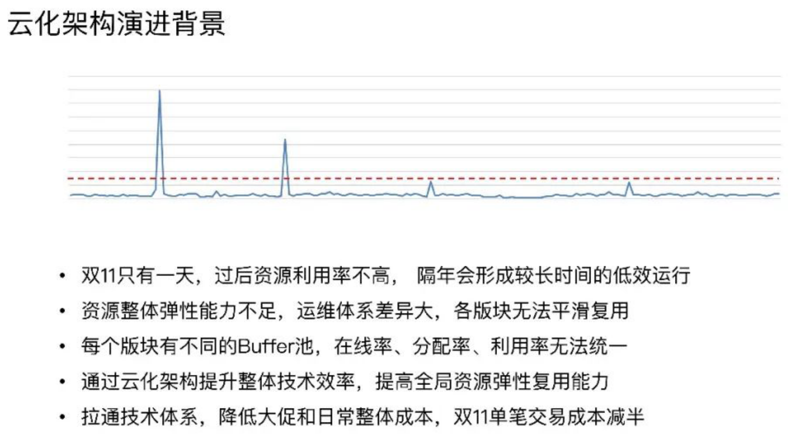
1. 互联网级规模，每天有数亿人在阿里网站上进行交易；
2. 企业级复杂度，每完成一笔交易都需要数百个系统的服务支撑；
3. 金融级的稳定性，每一笔交易都必须保证其完整性和正确性；
4. 双 11 存在数十倍的业务峰值，要求系统绝对稳定。

随着分布式架构、异地多活、限流降级、全链路压测等技术的突破，扩展性和稳定性问题得到了很好的解决。系统架构伴随 9 年双 11 的发展一代一代向后演进，每一年都有很大的提高。08 年开始阿里巴巴着手系统由集中式架构变成分布可扩展架构，其中沉淀了大量互联网中间件技术。2013 年通过异地多活的架构演进，把阿里完整的交易单元部署到全国各个城市站点，实现了地域级水平扩展。这两种技术的叠加解决了整个双 11 扩展性问题。

由于分布式架构的演进，系统稳定性问题开始凸显、系统复杂度急剧上升、多个系统间协同出现问题。我们建设了限流降级、预案体系、线上管理管控体系。在 2013 年时做了双 11 备战的核心武器——全链路压测。它能对整个系统的依赖关系里跟双 11 有关的部分进行完整的用户级的线上大流量真实场景读写压测，使系统提前经历几次“双 11”，验证整个线上生产环境处理能力，及时发现问题并修复。**目前这些技术已经成为互联网行业的标配技术。**

云化架构的演进

由于双 11 本身峰值增长很快，当我们做好了系统的稳定性后，发现硬件、时间、人力成本的消耗很大。成本挑战的出现推动我们解决 IT 成本的问题，即服务器资源问题。首先来看云化架构演进背景。



上图为阿里业务六个月的峰值数据表。表中两个最大的峰值依次代表双 11 和双 12 的交易峰值，其他较小的峰值是日常交易峰值，红线代表日常准备系统服务器资源的处理能力。

在 13 年之前，我们采购大量的服务器资源以支撑双 11 流量高峰。高峰过去后，长时间低效运行产生很大的资源浪费，这是非常粗放的预算和资源管理模式。阿里的多种业务形态产生了多种集群，每个集群之间运维体系差异较大、各个板块无法互用、资源整体弹性能力不足导致双 11 无法借用这些资源。每个板块的资源池有不同的 buffer，每个资源池的在线率、分配率和利用率无法统一。我们通过云化架构提高整体技术效率和全局资源的弹性复用能力。例如某个不做双 11 的集群把资源贡献出来给双 11 的交易使用。由于云能提供双 11 正需要的弹性能力，所以我们也开始大量使用阿里来解决双 11 成本问题，通过拉通技术体系来降低大促和日常整体成本，提出通过云化架构来实现双 11 单笔交易成本减半的目标。

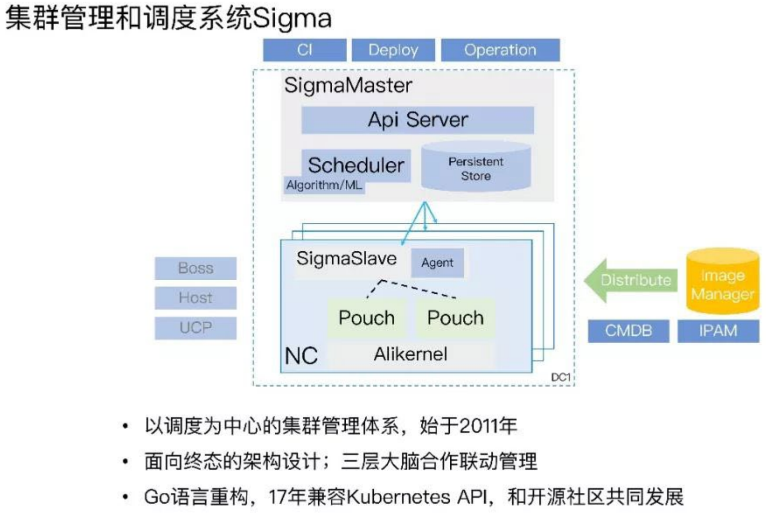
先来梳理一下整个运维体系现状。我们将集群大致分在线服务集群、计算任务集群、ECS 集群三类。这三种集群上的资源管理和基础运维、调度都是独立的。它们有各自的调度编排能力，在线服务 Sigma 调度、计算任务的 Fuxi 调度、ECS 的 Cloud Open API；它们在生产资源、资源供给、分配方式上也是不同的。在线服务用的是容器、计算任务调度最后生产的是 LXC 的轻量级隔离封装容器、云生产的是 ECS；它们在应用层上运维集群管理也是不一样的，最上层业务层跑的任务也不一

样。在线服务器跑的是在线业务如交易搜索广告、有状态的存储。计算集群跑的是大数据分析的任务，云集群跑的是各式各样的外部客户的任务。

通过技术全面云化逐层进行重构升级，建设弹性复用的能力实现全局统一调度。在线任务和计算任务混合部署，通过统一运维部署和资源分配的标准化提高调度效率，以此来实现容量的自动交付。所以我们需要做全面容器化，利用公有云，发挥云的弹性能力，减少自采基础设施的投入。通过混合云弹性架构和一键建站复用阿里云的能力，降低双 11 的成本，利用阿里云做到以前一年的资源持有时间缩减到只需 1–2 个月。

统一调度体系

始于 2011 年建设的 Sigma 是服务阿里巴巴在线业务的调度系统，围绕 Sigma 有一整套以调度为中心的集群管理体系。

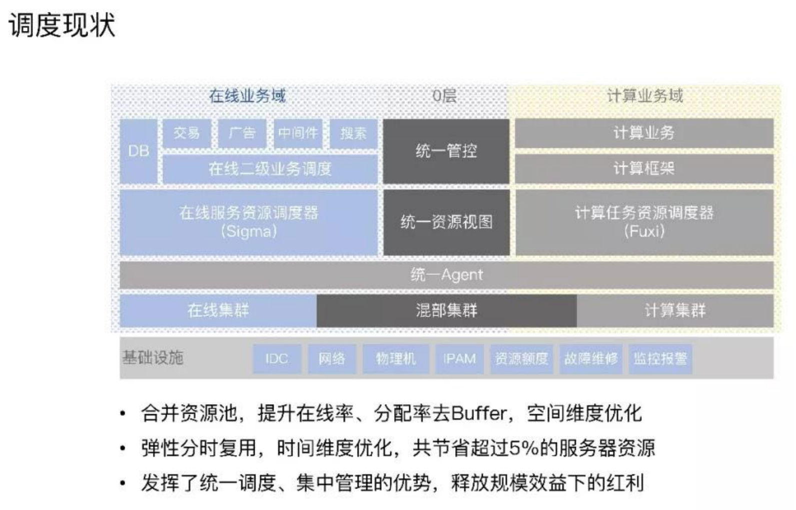


Sigma 是有 Alikenel、SigmaSlave、SigmaMaster 三层大脑联动合作，Alikenel 部署在每一台 NC 上，对内核进行增强，在资源分配、时间片分配上进行灵活的按优先级和策略调整，对任务的时延，任务时间片的抢占、不合理抢占的驱逐都能通过上层的规则配置自行决策。 SigmaSlave 可以在本机上进行 CPU 的分配、应

急场景的处理。通过本机 Slave 对时延敏感任务快速做出决策和响应，避免因全局决策处理时间长带来的业务损失。SigmaMaster 是一个最强的大脑，它可以统揽全局，为大量物理机的容器部署进行资源调度分配和算法优化决策。

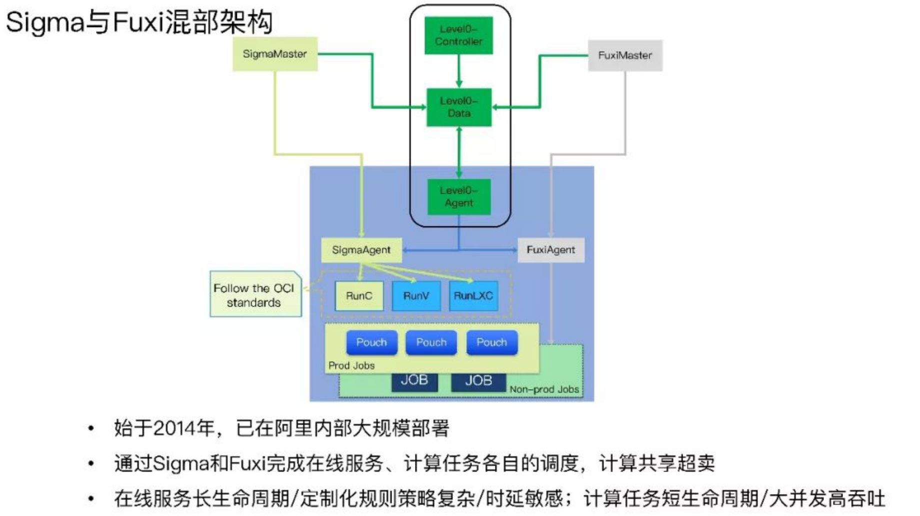
整个架构是面向终态的设计理念，请求进来后把数据存储到持久化存储，调度器识别调度需求分配资源。系统整体的协调性和最终一致性是非常好的。我们在 2011 年开始做调度系统，2016 年用 Go 语言重写，2017 年兼容了 kubernetes API，希望和开源社区共同建设和发展。

发挥统一调度，集中管理的优势，释放了规模效益下的一些红利。在线服务的调度和计算任务调度下有各种业务形态，它们在一层调度上进一步细分成二层调度，通过合并资源池提升利用率和分配率，合并 buffer 进行空间维度的优化实现全局打通。全局打通后进行弹性分时复用、时间维度的优化，共节省超过 5% 的资源。由于基数大，这个优化效果是非常可观的。



阿里巴巴在 2014 年开始推动混部架构，目前已在阿里巴巴内部大规模部署。在线服务属于长生命周期、规则策略复杂性高、时延敏感类任务。而计算任务生命周期短、调度要求大并发高吞吐、任务有不同的优先级、对时延不敏感。基于这两种调度的本质诉求的不同，所以我们在混合部署的架构上把两种调度并行处理，即一台 NC

物理机上可以既有 Sigma 调度又有 Fuxi 调度。Sigma 调度是通过 SigmaAgent 调用 OCI 标准的 RunC 、RunV、RunLXC 三种标准来启动 Pouch 容器。Fuxi 也在这台 NC 物理机上抢占资源，启动自己的计算任务。所有在线任务都在 Pouch 容器上，它负责把服务器资源进行分配切割通过调度把在线任务放进去，离线任务填入其空白区，保证物理机资源利用达到饱和，这样就完成了两种任务的混合部署。



混部的关键技术

内核资源隔离上的关键技术

- 在 CPU HT 资源隔离上，做了 Noise Clean 内核特性，解决在 / 离线超线程资源争抢问题。
- 在 CPU 调度隔离上，CFS 基础上增加 Task Preempt 特性，提高在线任务调度优先级。
- 在 CPU 缓存隔离上，通过 CAT，实现在、离线三级缓存 (LLC) 通道隔离 (Broadwell 及以上)。
- 在内存隔离上，拥有 CGroup 隔离 /OOM 优先级；Bandwidth Control 减少离线配额实现带宽隔离。

- 在内存弹性上，在内存不增加的情况下，提高混部效果，在线闲置时离线突破 memcg limit；需要内存时，离线及时释放。
- 在网络 QoS 隔离上，管控打标为金牌、在线打标为银牌、离线打标为铜牌，分级保障带宽。

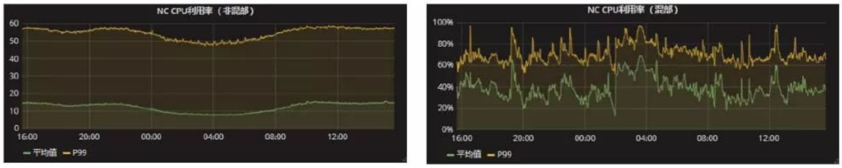
在线集群管理上的关键技术

- 对应用的内存、CPU、网络、磁盘和网络 I/O 容量进行画像，知道它的特征、资源规格是什么，不同的时间对资源真实使用情况如何，然后对整体规格和时间进行相关性分析，进行整体调度优化。
- 亲和互斥和任务优先级的分配，哪种应用放在一起使整体计算能力比较少、吞吐能力比较高，这是存在一定亲和性。
- 不同的场景有不同的策略，双 11 的策略是稳定优先，稳定性优先代表采用平铺策略，把所有的资源用尽，让资源层全部达到最低水位。日常场景需要利用率优先，“利用率优先”指让已经用掉的资源达到最高水位，空出大量完整资源做规模化的事。
- 应用做到自动收缩，分时复用。
- 整个站点的快速扩容缩容，弹性内存技术。

计算任务调度 +ODPS 上的关键技术

有弹性内存分时复用、动态内存超卖、无损降级与有损降级三个关键混部技术。动态内存超卖指内存是可以超卖的，如果有在线任务要用，就快速归还。有损降级和无损降级的策略指的是对影响在可接受范围内的波动干扰进行无损降级，不增加新任务，慢慢把它降下来，对影响大的干扰直接杀掉任务属于有损降级。利用零层管控，管理每台 NC 上的在线任务和离线任务之间的关系。

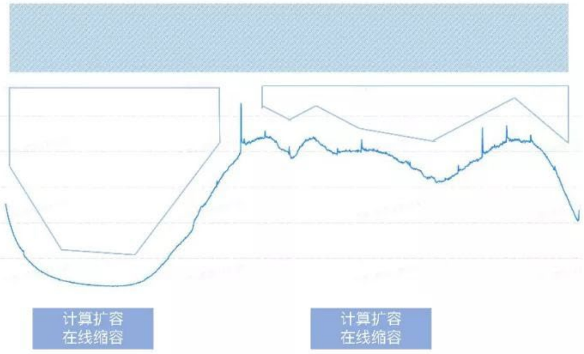
混合部署–引入计算任务提升日常资源效率



- CPU平均利用率10% –> 40%，延迟敏感类应用RT影响<5%
- 混部集群规模数千台，经过交易核心链路双11大促验证
- 为日常节省超过30%的服务器，明年会扩大10倍部署规模

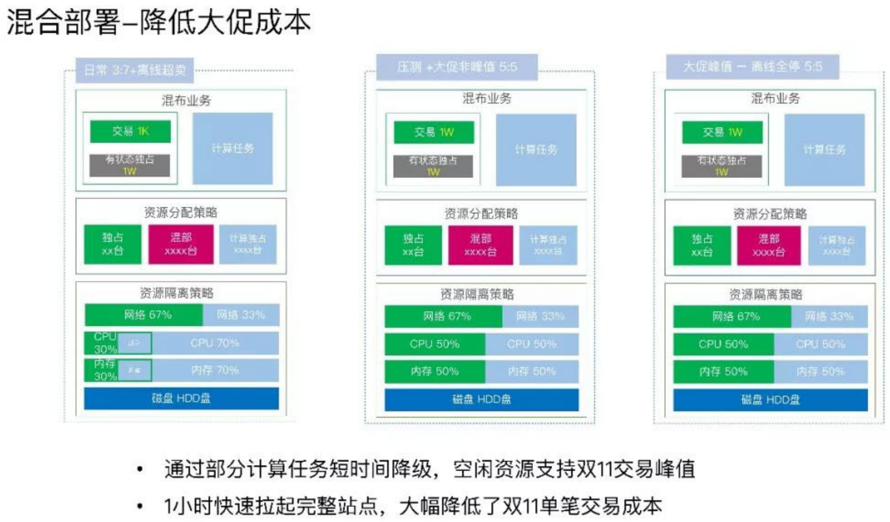
混合部署指将计算任务引入在线服务集群以提升日常资源效率。把离线任务引入后，CPU 平均利用率从 10% 上升到 40% 以上，同时敏感类应用的延迟影响时间小于 5%，属于完全可接受范围。目前我们整个混部集群已达到数千台的规模，经过了交易核心链路双 11 大促的验证。这项优化可以为日常节省超过 30% 的服务器。由于涉及到硬件的迭代和网络的迭代，需要一个很长的准备周期，所以我们预计明年会扩大 10 倍的部署规模。

混合部署–分时复用进一步提升资源效率



- 时间空间维度优化
- 结合弹性分时复用，平均CPU利用率提升至60%以上

通过分时复用，达到进一步提升资源效率的效果。上图中的曲线是我们某个应用的流量曲线。它是非常有规律的 , 左边代表晚上波谷期，右边代表白天处于波峰期。正常的混部指占用图中蓝色阴影部分的资源把利用率提高到 40%，弹性分时复用技术是指对应用画像找到应用流量波谷期，对应用缩容，大量释放内存和 CPU，调度更多计算任务。通过这项技术，把平均 CPU 利用率提升到 60% 以上。



在双 11 时，如何利用计算任务集群混合部署助力双 11 降低成本？我们把计算任务集群分成三种状态：完全没有在线服务的状态、在线服务和计算任务共同存在的过渡状态、双 11 时在线服务占主流计算任务短时间降级状态。集群混合部署后资源分配三七开，计算任务可以抢占在线任务分配的资源；在压测和大促非峰值时资源分配五五开；在大促峰值到来时，计算任务短时间降级，空闲资源支撑双 11 峰值。通过一小时快速建站拉起完整交易站点，大幅度降低了双 11 的单笔交易成本。

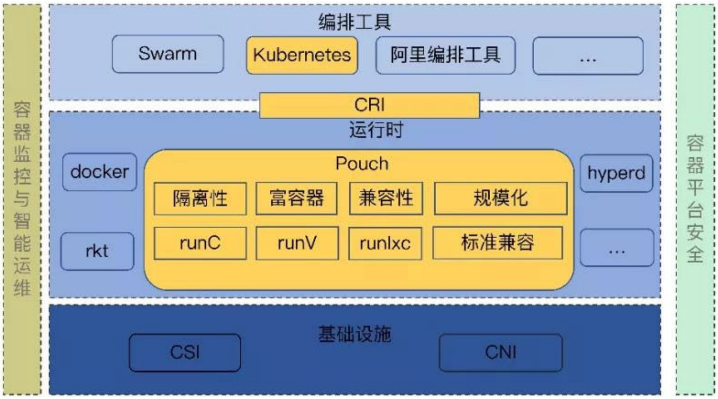
Pouch 容器和容器化的进展

全面容器化是提升运维能力、拉通运维差异的关键的技术。首先介绍一下阿里巴巴内部容器技术产品 Pouch。它从 2011 年开始建设和上线，基于 LXC，在 2015 年初开始吸收 Docker 镜像功能和很多标准。阿里巴巴的容器非常有特点，它结合了

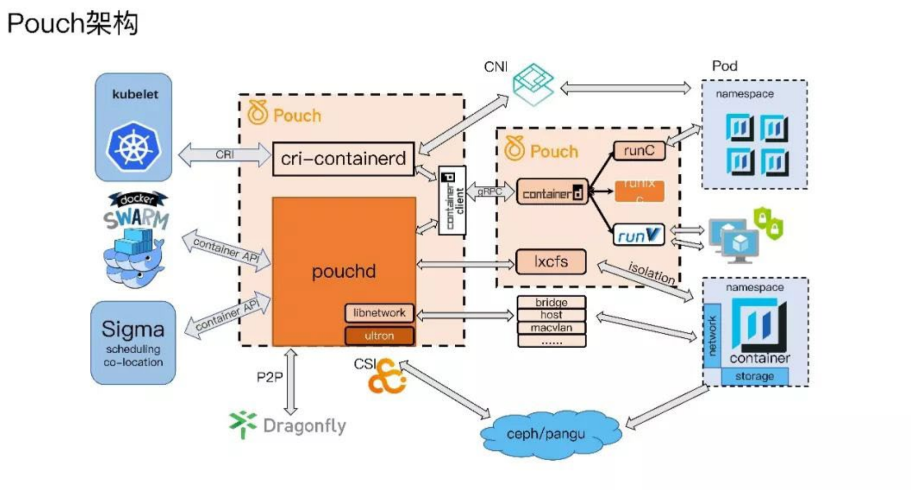
阿里内核，大幅度提高了它的隔离性，目前以百万级规模部署于阿里集团内部。

再来了解一下 Pouch 的发展路线。以前用的是虚拟机的虚拟化技术，虚拟化技术过渡到容器技术面临着很多运维体系的挑战。运维体系的迁移是一个很大的技术成本。我们做到了阿里内部运维和应用视角，有独立 IP，能够 ssh 登录，有独立的文件系统和资源隔离使用量可见性。2015 年以后，阿里巴巴引入 Docker 标准，形成了新的一套容器 Pouch 并集成整个运维体系。

Pouch定位



Pouch 的隔离性非常好，是富容器，可以登录容器，看到容器内进程自己占的资源量，有多少进程，进程挂了容器是不会挂的，可以运行很多的进程。兼容性很好，旧版本和以后的版本都支持，对利旧很有帮助。同时经过了百万级容器部署的规模化验证，我们研发了一套 P2P 镜像分发机制，大幅度提升分发效率。同时兼容了业界更多标准，推动标准的建设，支持 RunC 、RunV 、RunLXC 等标准。

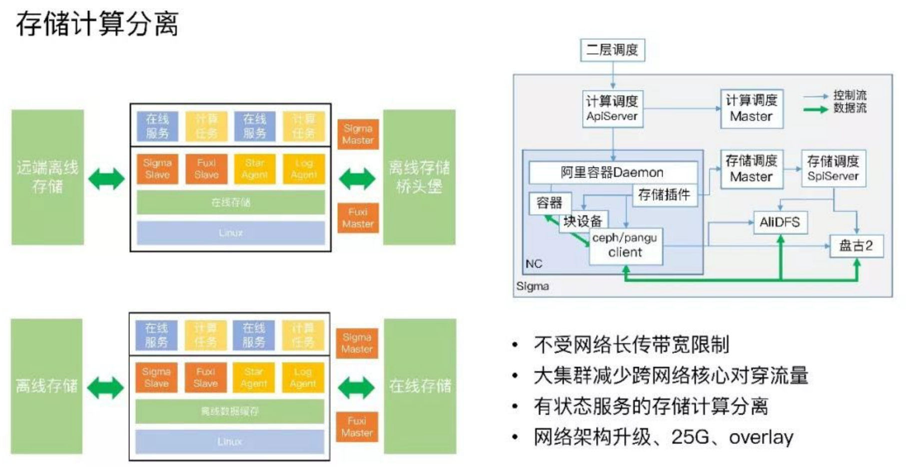


Pouch 的结构是比较清晰的，Pouchd 如何跟 kubelet、swarm、Sigma 交互。在存储上跟业界一起建设了 CSI 标准。支持分布式存储如 ceph、pangu。在网络上使用 lxcfs 增强隔离性，支持多种标准。

目前 Pouch 化覆盖了阿里的大部分 BU，2017 年达到百万级部署，在线业务达到 100% 容器化，计算任务也开始容器化，它拉平了异构平台的运维成本。覆盖运行模式，多种编程语言，DevOps 体系。Pouch 覆盖了阿里几乎所有业务板块如蚂蚁、交易、中间件等等。

Pouch 于 2017 年 10 月 10 号宣布开源，11 月 19 日正式开源，计划在 2018 年 03 月发布第一个大版本。我们希望通过 Pouch 的开源推动容器领域的发展和标准的成熟，给业界提供差异化有竞争力的技术选择。不仅方便传统 IT 企业利旧，老的基础设施也同样能够享受容器化带来的运维层的好处和优势，而且方便新的 IT 企业享受规模化稳定性和多标准兼容性带来的优势。

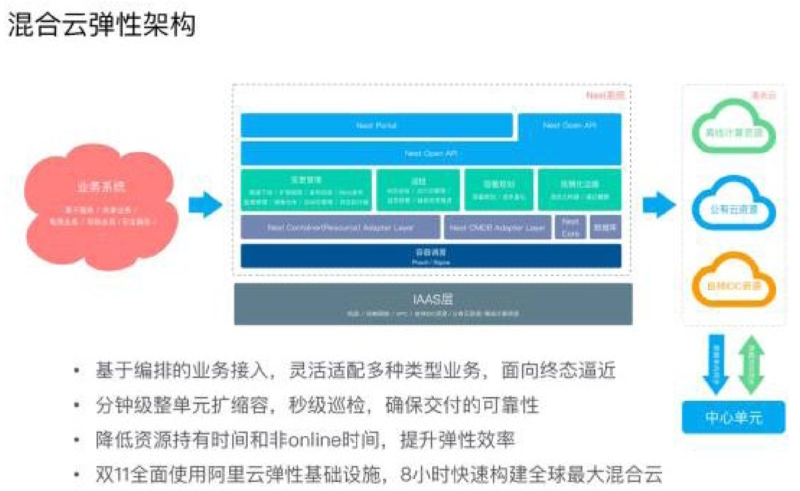
Pouch 开源地址：https://github.com/alibaba/pouch



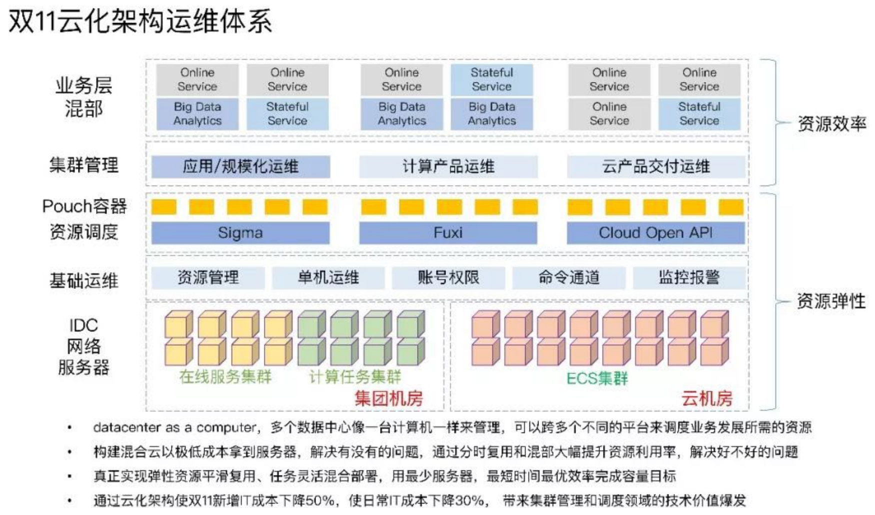
在存储计算分离上，由于有状态任务需要复制状态，会严重影响分时复用运维自动化程度和调度效率。我们在云化过程中实现了存储计算分离技术。由于计算集群和在线服务不在一个机房，计算任务的数据需要先缓存到在线业务的集群上，所以我们搭了一个缓存桥头堡，然后进行计算。

随着机房结构的调整和网络的优化，我们开始对在线计算同时进行存储计算分离。目前已经实现去桥头堡方案，不受网络上传带宽的限制，减少了大集群跨网络核心对传流量，提升了调度的灵活性。存储计算分离技术不仅可以使用阿里盘古技术，同时也兼容业界这一套容器上的存储标准，这也是阿里实现云化架构的非常关键的技术。在网络架构的升级上，我们大规模使用 25G 网络，在公有云上使用 VPC，overlay 能云上、云下和数据集群整个网络打通，这也是阿里大规模混合部署的前提。

云化架构和双 11 未来技术路线



这是阿里混合云弹性架构，是基于编排的一套技术体系，也是面向动态的架构。它可以分钟级进行单元的扩容和缩容，快速在云上或大数据集群上建立交易单元，实现秒级巡检，确保交付的可靠性。这套体系能持续降低资源持有时间和服务器的非 ONLINE 时间，降低损耗时间，提升了弹性的效率。双 11 超过 60% 的峰值流量都跑在阿里云上，全面使用阿里云弹性基础设施，8 个小时快速构建全球最大混合云。

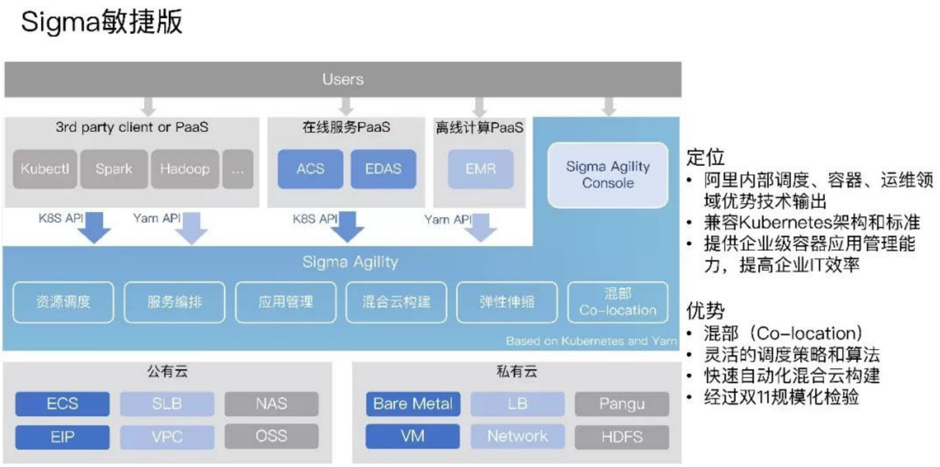


双 11 云化架构运维体系介绍

将资源分为在线任务集群、计算任务集群和 ECS 集群。资源管理，单机运维、状况管理，命令通道、监控报警这类基础运维体系已经打通。在双 11 场景中，我们会在云上划出一个独立的区域与其他场景互通。在互通区域，Sigma 调度可以到计算集群服务器里申请资源，生产 Pouch 容器，也可以到 cloud open API 去申请 ECS，生产出容器的资源。在日常的场景中 Fuxi 可以到 sigma 里申请资源，创建需要的容器。

在双 11 场景中，利用应用和规模化运维在容器上构建大量在线服务，包括业务层的混合部署，每个集群都有 online service 和有状态服务及大数据分析。阿里云的独占集群也部署了在线服务和有状态的数据服务，做到了 datacenter as a computer，多个数据中心像一台计算机一样来管理，实现跨多个不同的平台来调度业务的发展所需要的资源。构建了混合云用极低的成本拿到服务器，解决有没有的问题。

先有服务器规模，再通过分时复用和混合部署来大幅度提升资源利用率。真正实现了弹性资源平滑复用任务灵活混合部署，用最少的服务器最短的时间和用最优效率完成业务容量目标。通过这一套云化架构，我们在双 11 实现了新增 IT 成本降低 50%，使日常 IT 成本下降 30%，带来了集群管理和调度领域的技术价值爆发，也说明容器、编排调度技术的流行是一种必然。



后期我们会通过阿里云平台把内部储备的技术对外输出。这套技术是内部调度容器运维领域的优势技术，具备调度、编排、应用管理、监控、混合云的快上快下构建、弹性伸缩、混合部署的能力，同时兼容 Kubernetes API，提供企业级容器应用管理能力，提高企业 IT 效率，进而提高企业竞争力和创新效率。混合部署和自动化混合云构建技术经过双 11 规模验证，成为一个非常成熟稳定的技术体系。在云上，我们和 ACS 、EDAS 、EMR 合作，提高了产品的完整性。

未来云化架构技术将走向何方？

在双 11 场景下利用技术解决了大促成本的问题，找到了一个正确的方向，通过长期建设和发展，有更大的优化效率的提升。未来我们希望通过云化架构提升阿里 IDC 资源利用率、通过扩大调度规模和混部形态扩大效益。继续推进面向终态的体系结构和运维体系的提升，资源持有时间优化 30% 以上，持续降低大促的交易成本。

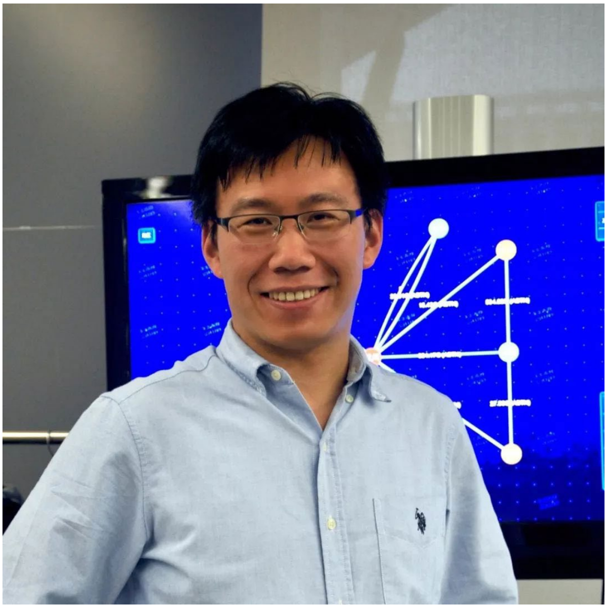
对双 11 本身，成本的问题已经得到了比较好的优化，未来着手于效率的提升，减少时间和人力成本，通过双 11 技术变量的采集、分析、预测微观视角的剖析和数据算法驱动，用智能决策进行处理。通过数据化、智能化、人与机器智能协同指挥，提升双 11 准备和作战效率，减少人力投入。通过加速基础技术的迭代，在体验、效率、成本和最大吞吐能力上找到新的平衡点，为整个行业和消费者带来一个更加完美的双 11。

阿里双 11 网络技术揭秘：百万级物理和虚拟网络设备的智能化之路

后羿

阿里妹导读：今天继续为大家分享 2017 双 11 背后不为人知的技术。这次的嘉宾是后羿，阿里巴巴资深技术专家，参与 8 年双 11 大战，主导阿里“去 IOE”战略落地，目前在推动阿里基础设施智能化。

后羿将分享双 11 的智能化网络实践，关于如何在网络智能领域通过数据手段极致地优化运营场景，在稳定性、成本、效率方面提升网络运营竞争力。



阿里巴巴资深技术专家后羿

后羿：大家好，首先给大家呈现的是阿里巴巴在双 11 中主要依赖的网络相关技术。在今年双 11 中我们在稳定性、高性能网关、去堆叠以及 25G、骨干网流量调度平台、流量的精准评估、QOS 优化和成本优化方面都取得了突破性的进展。

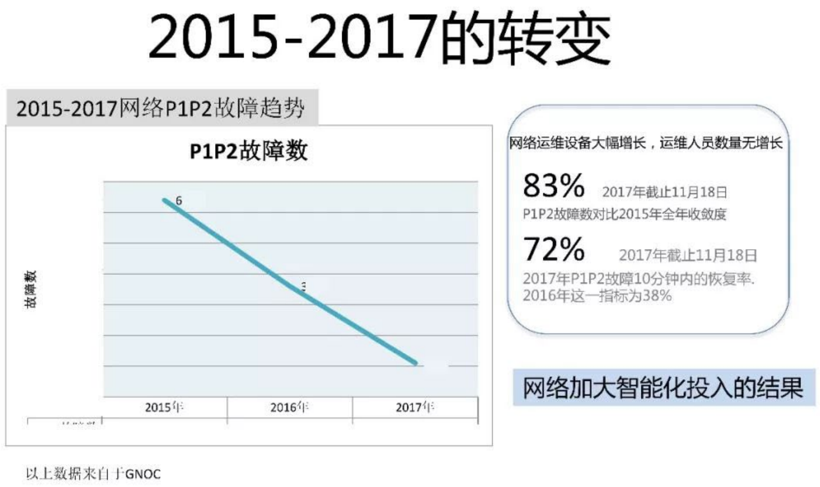
助力双 11 的重要网络技术

在稳定性的强化方面，在过去一年，阿里巴巴借助智能化手段，在故障的快速发现、自动修复、自动变更、快速诊断的能力上都有很大的强化，使之轻松应对双 11 流量高峰时的突发稳定性问题；在优化高性能网关方面，ANAT 吞吐量性能 16 倍的提升和 LVS 性能 8 倍的提升使阿里巴巴轻松应对 ANAT 转发网关的压力；4.2 架构支持去堆叠能力，提高了架构可靠性；25G 技术在 5.0 网络架构开始规模化使用，在存储计算分离和在线混布场景中也开始落地；骨干网流量调度平台做到了保底带宽、延迟的优化等方面都有好的用户体验。

在过去一段时间内，阿里云水立方做到了基于应用维度、按时间维度、任意角度的灵活运营流量精准评估能力。利用水立方预测双 11 业务流量和容量的分配，在端

到端 QOS 优化方面，阿里巴巴在存储计算分离，在线离线存储混布场景，及交易、支付等对用户体验要求较高场景中获得了更好的用户体验，保证相关的请求能得到优先的传输。在成本优化方面，AGN2.0 骨干网升级取得了很大的进展，自研光模块和 AOC 的全面落地都使得整体成本得到很好的优化。

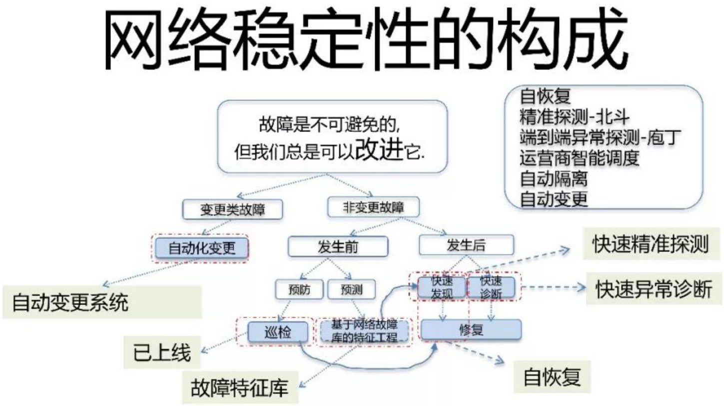
阿里巴巴是一个拥有百万级物理和虚拟网络设备、承载多样业务的遍布全球的统一的物理网络。不同的供应商在不同时期、不同版本、不同架构的管理都是不同的，我们需要付出更多的精力驾驭一个复杂的网络结构。面对大量级的物理和虚拟网络设备时，如何用一套优化的工程方法去进行分析数据；如何基于这些数据在后期做快速故障发现和定位；不同形态的业务对网络有不一样的需求，如何在兼顾资源利用率同时达到用户体验很好的平衡；在面临业务波动频繁的情况下，如何自证清白；在这些过程中如何快速完成综合处理……这些都是阿里巴巴需要解决的客观的工程难题。



上图呈现的是我们在 2015 年之后在网络稳定性提升方面的具体数据。从这张图中可以看出，我们在 15 年到 17 年期间，稳定性得到了很好的优化。2017 年 P1 P2 故障数对比 2015 年全年收敛了 83%；P1 P2 的故障数在十分钟内的恢复率对比 2016 年也得到了很好的改善。2016 年在 10 分钟内的故障恢复率为 38%，而在 2017 年则达到了 72%。需要强调的一点是，阿里巴巴网络设备大幅度增长，而网络

工程师和网络运维人员并无大幅度增长。这主要得益于过去两年我们在智能化上的投入。

如何改进处理故障过程?



我们将网络运营中的故障简单的划分成变更类故障和非变更类故障：

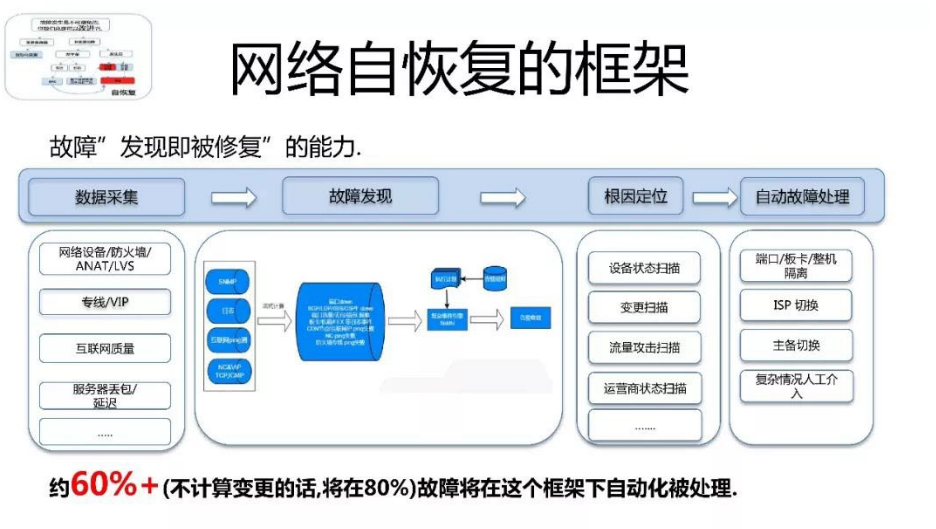
- 对于变更类故障，借助自动化变更这类自动化工具来解决变更带来的稳定性隐患，通过快速迭代、快速优化过程让故障快速收敛。
- 对于非变更类故障，在故障发生前，通过加大巡检力度，实时探测当前线上的配置是否存在漏洞，并将巡检结果呈现给运营工程师，运营工程师会系统化的逐步修复这些漏洞。

我们也在构建科学预测方法，用网络故障库的形式逐步构建全网网络故障特征工程。利用特征库预测故障存在的可能，做到防范于未然。在故障发生后，做到快速发现、快速诊断，当我们已经可以很好的定性一个特征故障时，快速对其进行修复。

快速发现模块主要是用来提升精准探测能力，诊断模块用于提升端到端故障诊断速度。同时，我们也在积极构建整体网络故障特征库。通过分析历史网络故障体现的量化特征，精确描述故障的形态和量化特点，帮助我们预知未来网络的潜在的故障。巡检系统在过去一年已经稳定上线，自动化变更系统帮助我们很好的驾驭每一天面临

的大量的变更需求。这些就是我们在解决网络稳定性方面的整体思路。

当我们已经可以发现故障、定性故障时，通过监控系统和修复系统的快速联动完成自修复，从而达成闭环，这就是阿里巴巴网络故障的自恢复。下图展示了网络自恢复过程及其自动完成信息的对接和中间逻辑的判断。



网络自恢复相当于快速发现和修复两个模块的自联动的过程。当故障已经发生时，如何做到“发现即被修复”？

网络自恢复主要有以下五部分构成：

- 端口 / 链路类异常自动隔离。
- 板卡类异常自动隔离。
- 运营商流量智能调度容灾切换。
- 堆叠分裂类异常自动恢复。
- 防火墙异常的自动切换。

后续会逐步加入更多的场景。随着场景的增多，到目前我们已经有 60% 以上的风险隐患实现了自动化的处理，大大降低了故障问题处理的时长，实现了真正的故障快速恢复，这也证明我们全面进入了自动化调度的时代。网络故障处理全面进入自动

化处理和智能化调度时代，60% 以上的风险隐患已经实现了自动化处理，大大降低了问题处理时长，实现故障的快速恢复。

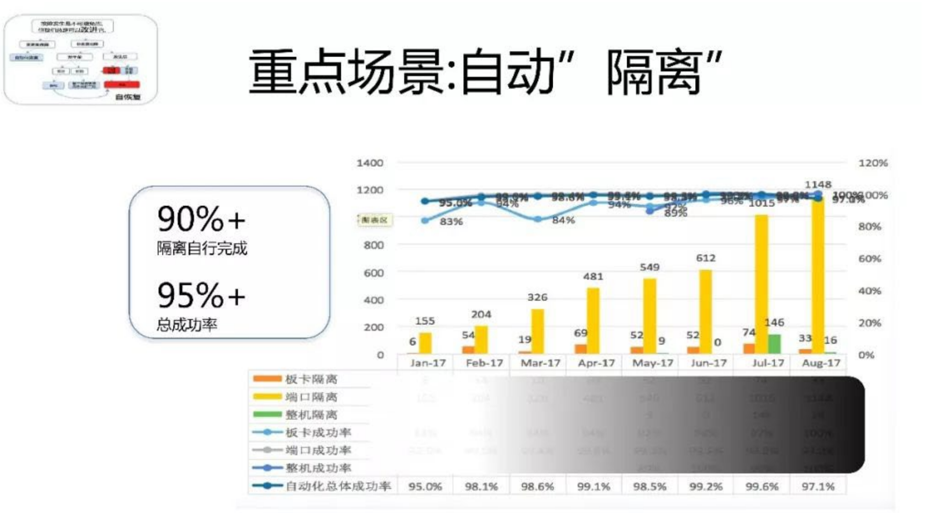
自恢复是一种怎样的体验？当监控系统探知到一个具体故障正在发生时，就会调用修复模块来完成故障修复，并在发现故障和修复完成故障后推送一条信息告知用户情况。这个过程几乎不需要人为的干预。我们希望借助一个大脑全面评估当下稳定性的情况，精准确认问题后通过调度工具平台完成修复过程。这也是一个推动智能化的过程。

智能调度与自动隔离

如何解决好运营商的割接以及网络的抖动的问题，避免用户体验的下降和故障的发生是我们花很大时间研究的课题。通过对网络质量的全面感知，告诉业务系统哪里正在出现网络质量恶化和变动，这意味着我们需要做一些工作来改善整体用户体验。在实际操作过程中，有很多细节需要我们考虑。运营商自动切换的过程基本都能在不需人工干预的情况下快速完成。

从图中可以看出，自从上线了自动化场景后，BGP 出口自动化切换的成功率是 100%，每自动化切换一次都意味着系统帮助我们规避了一起故障。

在自动隔离场景中，由于网络设备在运行过程中经常会出现故障，在快速修复之前前，隔离是在网络工程师解决问题的首要工作。从图中可以看出，自动隔离功能上线后，90% 以上的隔离操作能自动完成，而且成功率高达 95%，这样不仅省去了很多的人工还规避了很多潜在故障。



以上数据来自 GNOC

基于北斗系统的“快速发现”

北斗故障识别智能引擎有在线日志实时分析、异常流量实时探测、告警收敛三大模块帮助精准定位和发现。在线上我们每天要处理万亿级的数据信息，通过算法识别出大概 1 亿条的基础事件，进一步识别后我们形成 23 万左右的复杂事件，对复杂事件收敛形成 300 条左右的事件，其中有进 30 条左右被转化为工单。工单一般是通过人工干预或无人值守自动化方式消化工单。

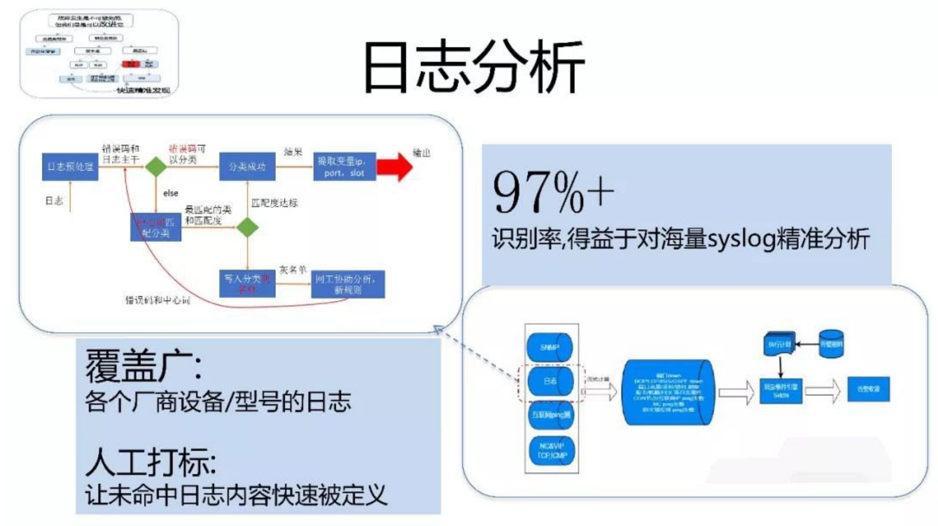
北斗故障识别智能引擎的工作流程主要分为四步：

- 利用庞大的数据采集系统，将 N 多维度数据实时从设备服务器中采集汇总；
- 在实时计算平台中利用各种机器学习算法和领域规则来完成基于场景的综合分析；
- 通过各种告警规则生成复杂事件；

- 对复杂事件进一步收敛。

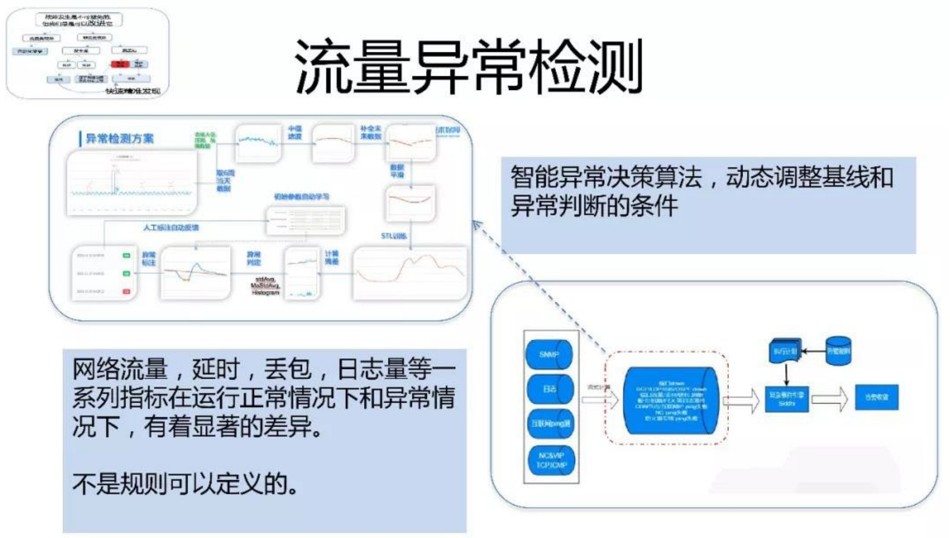
在线日志实时分析

我们已经对海量实时日志有 97% 以上的识别率，每天处理数亿条平面日志，从日志中通过文本分析和积累，加上人工打标，覆盖了所有厂商日志型号。剩余 3% 也有经验丰富的网络工程师帮助我们进一步打标，完善知识库。这是日志分析的大概运作原理。



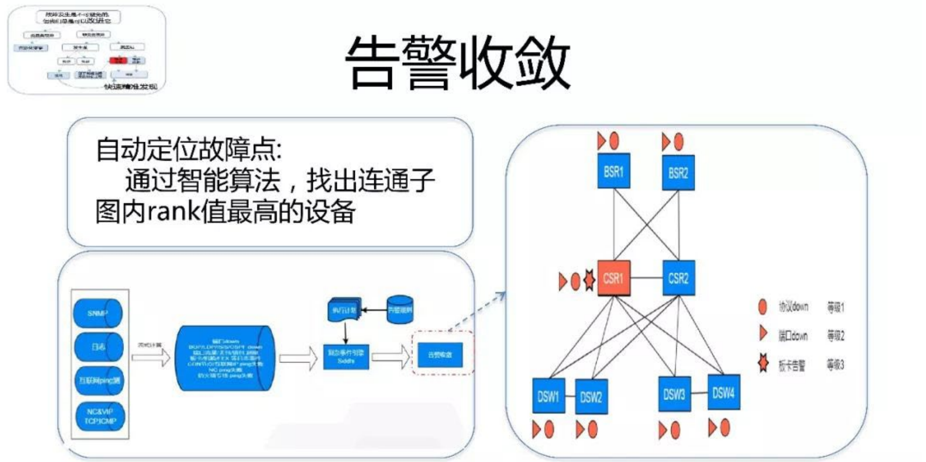
异常流量实时探测

为什么我们需要专门的模块来做异常流量的识别？因为某些数据不能通过传统方式确认其是否异常，如延迟、日志量、网络流量，这个数据在某个时段是正常的，但在另一个时段里是异常的。流量异常识别模块解决了如何构建一种智能决策算法，根据时间点和场景动态调整对应基线的问题。



告警收敛

当收敛出几十万条异常事件后，如何进一步确定异常的来源？我们将网络的拓扑加入在图计算引擎中。在对应一个时间窗口内，点亮所有产生告警信息的事件对应的拓扑图结构上。当连续一段拓扑被点亮后，把它当做一个故障联通子图，利用智能化算法对对应节点打分。通过 rank 值来确定出现故障设备源头。



动变更的作用

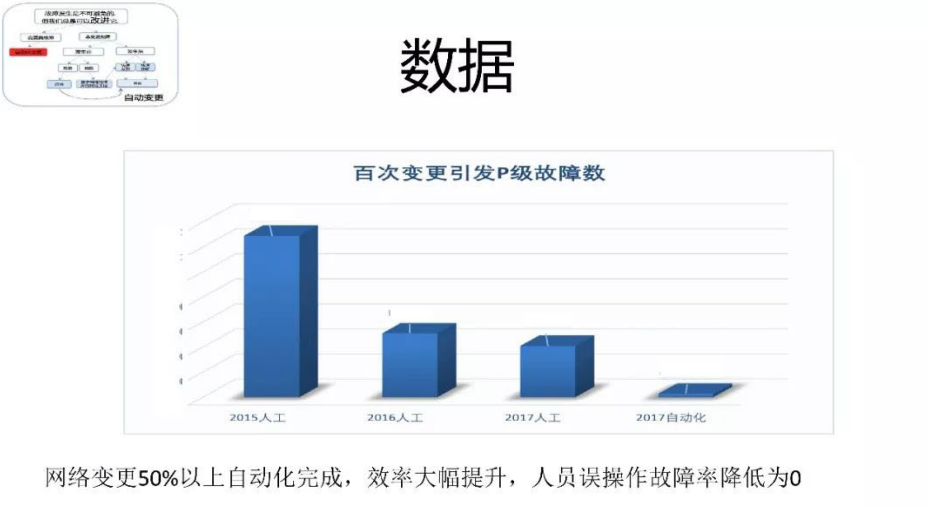
自动化变更已经成为一个非常基础的能力，它和内部很多工具模块和业务平台完成对接，使数据得到了打通，降低故障率的同时提高效率。

为什么要有自动变更模块？

1. 在运营百万级网络设备的情况下，每天会面临非常多类似打补、OsS 升级、路由变化、IP 扩容、回收等的变更需求。
2. 在过去，这些变更操作高达 85% 的部分都是由人工来完成的。有些业务的操作需要规避白天时间，很多工程师由于长期在晚上进行高危变更操作，得不到好的休息，工作容易出错导致性循环，带来难以控制的风险。
3. 由于变更工作的线下操作，很多可以变成经验的东西没有很好的在线上沉淀，而线下监测环节又比较薄弱。
4. 历史上一边工程师在操作变更，一边故障在蔓延的事不仅一次出现。如何做到变更的同时进行监测，实时感受变更现场网络态势感知是非常重要的。
5. 一些高危的变更需要引入审核机制，这些都是我们之前面临的现实问题。

我们是如何解决上述问题的呢？

总的来说就是运用通用的方法，更多的引用智能的手段，减少人工介入。一块块简单的乐高积木可以拼凑出如房子、飞机等非常复杂的形象。乐高积木的例子启示我们对需要展开的变更操作进行原子化的抽象，然后运用状态机组合成各式复杂的变更。在变更的同时，实时采集对应设备线上的告警信息，这些信息能告诉我们当下的变更是怎样一种情况。变更进行过程中是否有大量告警信息急速蔓延，决定着我们当下是否需要回滚，是否需要做现场决策和支持。



从图中可以看出，在 2017 年自动化变更上线后，变更引起的故障率有很大的降低，50% 以上的变更实现了自动化，人员的误操作概率降为 0。可想而知，变更的优化效率得到了很大的提升。

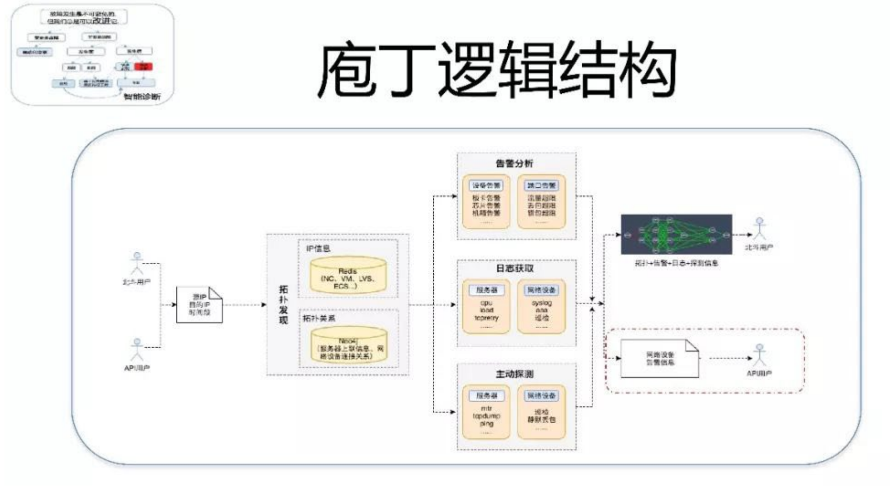
网络端到端智能快速诊断系统“庖丁”

在实际中我们经常会面临这样一个问题，某个地方丢包比较高或者两个点之间应用出现了严重的超时，究竟是怎么引起的？如果用人工的方式进行定位，首先要解决如何了解两个点之间端到端网络拓扑是怎样一种结构。拓扑上现在有故障在发生吗？如果有，这些故障设备究竟产生了哪些日志、过程中是否有变更在进行？如果已经知道是哪些设备为可疑对象，可能接下来对设备进一步下发命令、对数据做深入诊断，整个过程大概需要 1-2 小时。

而庖丁可以同时进行网络拓扑发现、告警信息自动聚合分析、日志信息自动获取、命令工具自动下发这四项工作，把整个复杂问题的定位时长从 1-2 个小时缩减为 3 分钟，给各类场景带来极大的诊断效率提升。针对已经确定的两个点的 IP，我们自动定义出所对应的 IP 拓扑是怎样一种结构；对相应拓扑链路上的所有日志进行实时提取、标注关键词；对可疑设备的告警进行自动化聚合收敛、过滤无效信息；主动对可疑设备进行可疑探测、做二次分析。这些过程几乎是一键完成。

庖丁运作的可视化呈现如图。对可疑故障链路进行标红处理，通过庖丁可视化界面，轻松判断故障的发生原因。

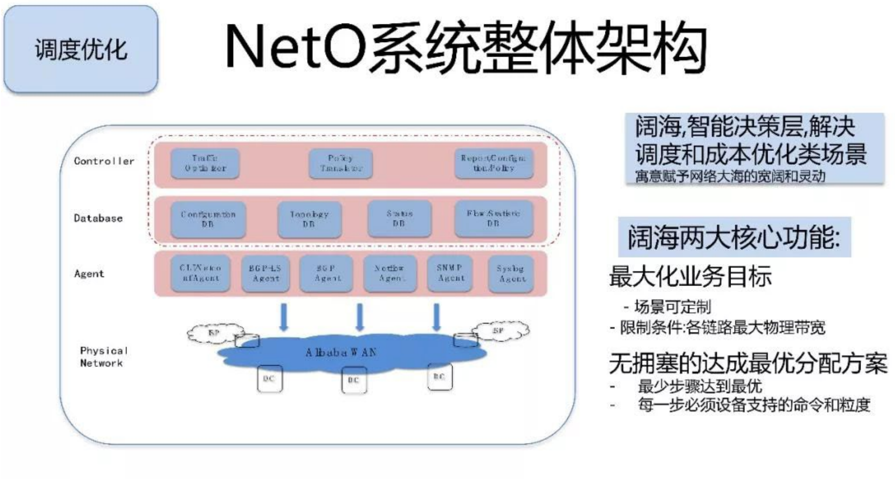
在故障发现、探测的最终结果可以对具体的用户呈现，也可以通过 API 形式对业务系统进行主动的信息推送。这意味着上层业务网络查询更加开放，通过对庖丁的一次查询可以得知某个业务波动是否是属于网络带来的问题。



基于 NetO 做流量最优化的分配

通过最优化流量分配来榨干多余带宽成本，同时满足最优路径选择、带宽扩容、稳定性方面的现实需求。

- 技术层面。我们希望每次网络路径都是最优的。传统网络基本基于 Metric 机制确定最短路径。对于阿里这张具有多样链路的网络，交易链路对网络的延迟极其敏感，大数据需要很大的带宽，需要更多可达路径帮助快速进行数据的传输。
- 带宽扩容角度。在面临非常频繁的带宽扩容需求情况下，实际的定时链路存在很多延时差异，两个点之间的路径带宽差异也很明显，我们需要站在运营的角度构建某种方法，既能充分利用闲置的带宽，又能在调配流量过程中很好的兼顾时延和成本。
- 稳定性方面。并行的链路在出现单点故障时，需要对其进行隔离，隔离后如何触发高可用路由决策。这些都是 NetO 需要解决的问题。NetO 基于 SDN 采用了 SR-TE 技术，帮助我们在全局情况下拿到全网流量信息、路由状态信息，用这些信息帮助我们按场景进行路径转发。



NetO 整体智能决策层模块——阔海

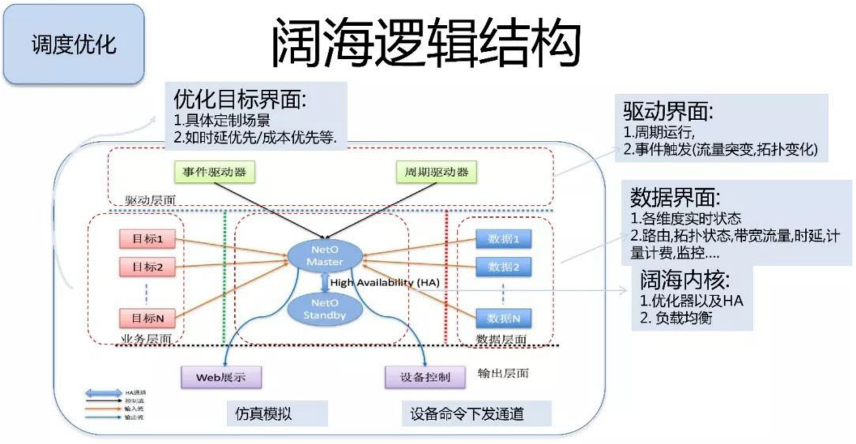
阔海有两大核心职能：

- 最大化业务目标。不同的场景有不同的需求，我们希望 NetO 可以根据各种限制条件对每个场景综合分析，定制最优解决方案。
- 以无拥塞方式达成最优分配方案。这要求我们最少的步骤解决问题，每一步对应的命令需要设备的支持。阔海帮助我们做到最大化利用链路上限，在每次流量调整中，即不触及带宽上限又能完成最优化调整，实现最小步骤的迁移。

阔海有两种驱动方式，一是周期性运行；二是通过突事件触发，如拓扑发生变化、流量发生变化等。阔海一个数据平台，需要用各个维度的实时数据来进行现状态势感知，通过数据背后业务含义帮助我们制定最优化分配方案。这些方案完全可以按不同需求对成本、时延、带宽利用率组合定制场景。

阔海有非常好的可靠性来帮助它做负载均衡。每次计算出的最优化结果可以通过两种方式来呈现：

- 通过仿真在 web 页面来呈现，告诉运营决策人员最优化结果会达成怎样的效果，让对应运营人员做现状评估。
- 直接用最优化结果进行设备命令的下发，完成一次优化调度。



这里给大家举三种常见的场景，黑色线条代表物理链路，其他颜色线条代表逻辑链路。

故障状态下的负载均衡

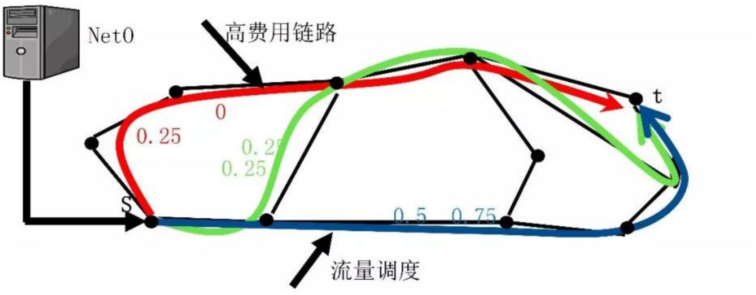
从第一个场景的图中可以看到三条链路在初始状态下进行数据的通信。通信链路出现单点故障时，NetO 会把蓝色链路的流量动态的分配到其他两条链路上去。

调度场景1:故障状态下，负载均衡

针对高费用链路的解决措施

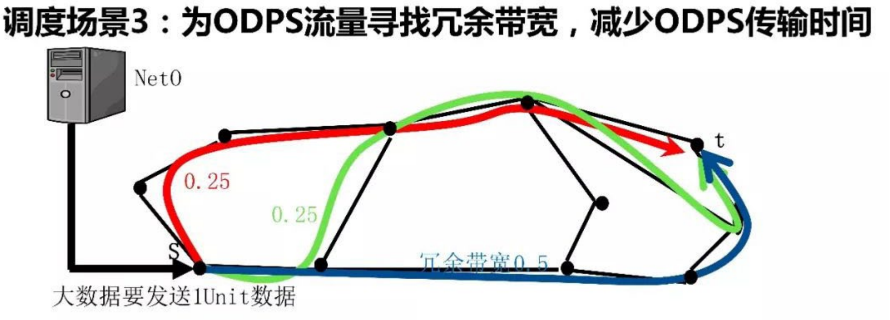
从实际角度出发，每条链路意味着不同的资费，为了节省成本，提高资源利用率，我们完全可以采取灵活的策略来运行。如下图所示，我们在运行过程中发现其中一条链路的成本偏高，这时 NetO 会自动触发一次调用，把流量分配到相对来说成本较低的链路上，这个过程基本不需要人工的干预。

调度场景2：用户体验不变，节省资费



大数据场景优化传输时间

比如我们需要发送一个单位的数据，在初始状态下，以图中红绿两条链路发送数据时，由于带宽较小，需要两个时间周期完成数据的传输。NetO 在整体链路上找到另外一条冗余带宽（蓝色链路），并提示系统把这个链路利用起来，这个调度过程触发了流量的再次优化分配。原本需要两个时间单位传输的数据在这条链路上一个时间单位就能完成。



以上就是阿里巴巴在双 11 中的网络智能化技术及在成本优化、流量智能化调度等方面相关实践的介绍。网络智能永远是一个在路上的过程，我们还在不断努力演进它。在未来一段时间内，我们会进一步在无人值守、成本优化和稳定性方面加大投入，给大家呈现更好的东西，带来更好的用户体验。

从 10% 到 40%：阿里巴巴混部技术权威详解

潇谦

背景引言

每年双十一创造奇迹的背后，是巨大的成本投入。为了完成对流量峰值的支撑，我们需要大量的计算资源，而在平时，这些资源往往又是空闲的。另一方面，为了在极端情况下，如机房整体断电等还能保障阿里巴巴的业务不受损失，也需要在全国各地建立冗余资源。而且就算是一天当中，在线服务的负载也是不一样的，白天一般情况下要比凌晨高得多。根据盖特纳和麦肯锡前几年的调研数据，全球的服务器的 CPU 利用率只有 6% 到 12%。即使通过虚拟化技术优化，利用率还是只有 7% - 17%，而阿里巴巴的在线服务整体日均利用率也在 10% 左右。

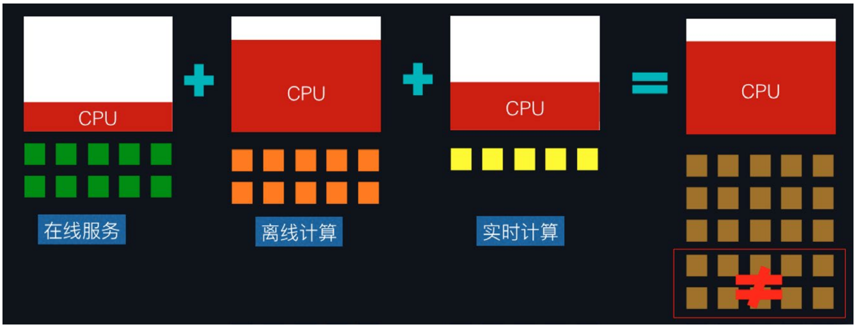
另一方面，全球从 IT 时代全面走向了 DT 时代，现在又在向更深入的 AI 时代迈进。各各样的大数据处理框架不断涌现，从 Hadoop 到 Spark，从 Jstorm 到 Flink，甚至包括深度学习框架 Tensorflow 的出现，成千上万的数据分析背后是大量的计算任务，占用了大量的计算资源。由于计算任务占用的计算量很高，CPU 水位通常在

50%–60% 以上，不同于在线服务，计算任务的峰值通常出现在凌晨，水位甚至能达到 70% 以上。所以我们往往就会建立独立的计算任务集群。

很多人都被车堵过，而堵车的时候，并不是所有的车道都在堵车。有一个比较有趣的情况，我们称之为潮汐现象，而它造成的问题是在早高峰的时候是进城方向堵车，而晚高峰是出城方向堵。而为了缓解这个问题，我们使用了潮汐车道的方式。

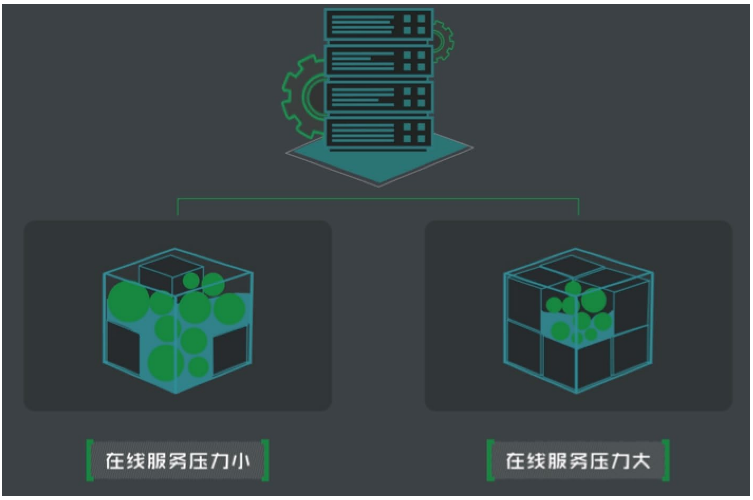
那么同样的原理，是否如果能让这两个集群混合起来部署，让计算任务的一部分任务跑到在线服务的资源之上，把在线服务空闲的资源利用起来呢？答案是肯定的。

混部技术简介



混部技术示意图

把集群混合起来，将不同类型的任务调度到相同的物理资源上，通过调度，资源隔离等控制手段，在保障 SLO 的基础上，充分使用资源能力，极大降低成本，我们称这样的技术为混部（Co-loaction）



打个比方，跑在容器里的在线服务就像石块；而计算任务我们把它比喻成沙子和水。当在线压力小的时候，计算任务就占住那些空隙，把空闲的资源都使用起来，而当在线忙的时候，计算任务就立即退出那些空隙，把资源还给在线业务。这样的技术一方面在平时，我们可以极大地提升资源的利用率；另一方面，在大促活动需要突增在线服务器的时候，又可以通过在线业务占用计算任务资源的方式，来顶住那短暂的峰值压力。

从原理中我们可以看到可以混部在一起的任务有两个比较重要的特征：

1. 可以划分优先级：一定需要优先级比较低的任务，它们能像水和沙子一样，随时能被赶走，而不会受到不可承受的影响，让优先级高的任务不受干扰。在线的特点是：峰值压力时间不长，对延时比较敏感，业务的压力抖动比较厉害，典型的如早上 10 点的聚划算活动，就会在非常短的时间内，造成交易集群的压力瞬间上升 10 几倍，对于稳定的要求非常高，在混部的时候，必须要保证在线的通畅，需要有极强的抗干扰能力。而计算任务的特点是：平时的压力比较高，相对来说计算量可控，并且延迟不敏感，失败后也可以重跑。至

少需要几分钟跑完的计算任务，相对于几秒甚至几十秒的延迟，并不会产生严重的问题，正好可以承担起水和沙子的角色。

2. 资源占用互补性：两种任务在不同的时间点对水位的占用不一样。如在线服务是，平时比较低，大促时比较高；凌晨比较低，白天比较高。而计算任务则反过来，平时比较高，大促时可以降级；凌晨非常高，白天却要低一些。

这种方式带来的成本节省是非常巨大的：假设数据中心有 N 台服务器，利用率从 R1 提高到 R2，不考虑其他实际制约因素的情况下，节约 X 台，那么理想的公式是：

N*R1 = (N−X)*R2

=> X*R2 = N*R2 – N*R1

=> X = N*(R2−R1)/R2

也就是说如果企业有 10 万台服务器，利用率从 28% 提升到 40%，代入上述公式，就能节省出 3 万台机器。假设一台机器的成本为 2 万元，那么节约成本就有 6 个亿。

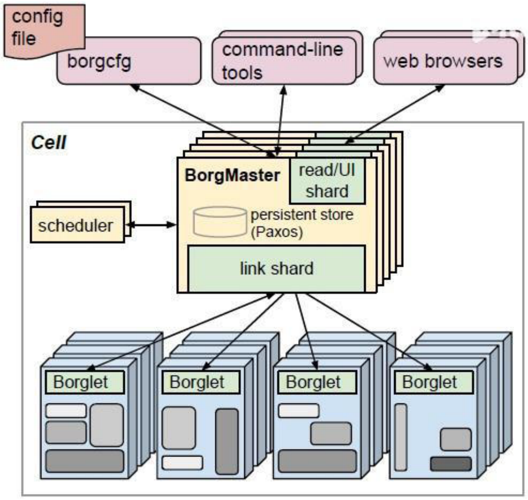
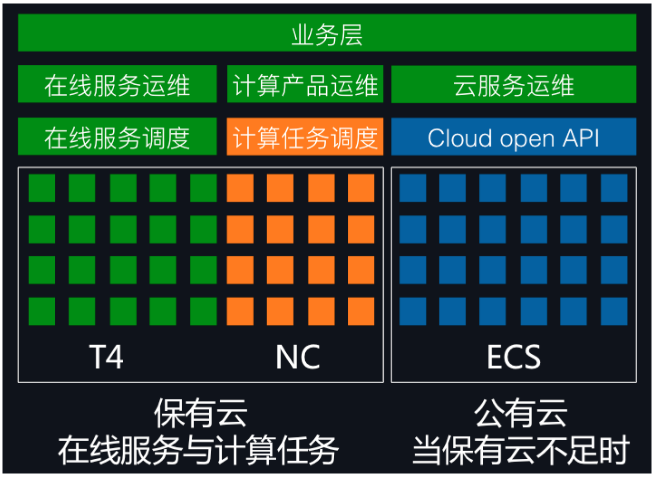


Figure 1: The high-level architecture of Borg. *Only a tiny fraction of the thousands of worker nodes are shown.*

2015 年，Google 发表了 Borg 论文，其中就提到了在线服务与计算任务之间的混合运行，也就是我们说的混部技术。Borg 论文中描述了 Google 由于采用了这项技术，为 Google 节省了 20%–30% 的机器规模

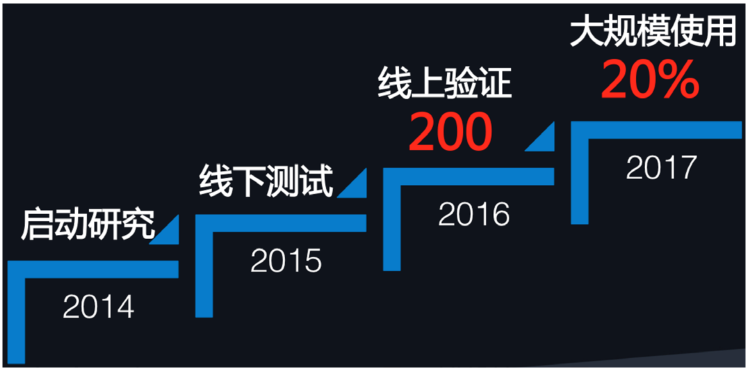
混部技术的历程



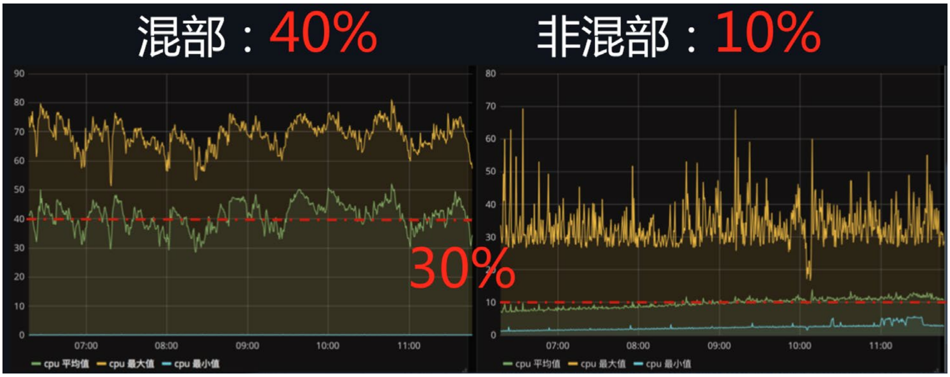
阿里巴巴早期混合云架构

大家都知道这今年阿里巴巴双十一的交易峰值是每秒 32.5 万比，相比去年几乎增加了 1 倍，但是这样的高峰却只有 1 小时左右。为了让交易的成本降低，从 2014 年开始，我们一方面通过阿里云的公有弹性云资源降低成本，另一方面也开始研究混部相关的技术。

混部能产生这么大的帮助，可是业界能使用在生产的没有几家公司，其原因也非常简单，第一个是规模，第二个是技术门槛。当你机器规模不够大的时候，显然意义不大。而在技术上，计算型任务通常都可以把利用率跑到很高，如果计算型任务和在线型业务运行在同一台机器上，怎么避免计算型任务的运行不会对在线型业务的响应时间等关键指标不产生太大的影响呢，这个需要在技术上有全方位的突破，而阿里巴巴从无到有，花了 4 年多的时间才让这项技术在电商域得以大规模落地。

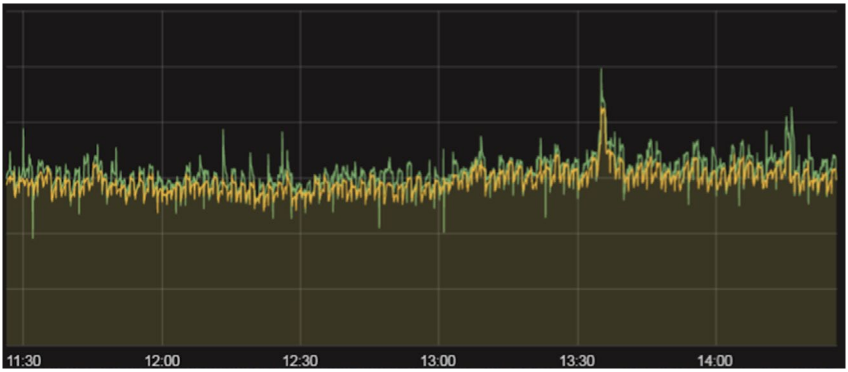


1. 2014 年，我们最主要的工作是进行技术论证，方案设计，以及相关的一些实验性研究
2. 2015 年，我们开始了日常测试环境的测试工作。这一期间让我们总结了相当多的问题：如调度融合问题、资源争抢隔离问题、存储依赖问题、内存不足问题等等。
3. 2016 年，当我们把大部分问题都解决掉时，我们开启了线上 200 台左右的小规模验证。由于电商的金融属性，对于抗干扰要求特别高，在不断的业务考验下，我们不停地修正着技术方案。
4. 2017 年，经过一年的磨合，混部的整体技术终于走向了成熟和大规模生产。阿巴巴双十一当中，约有 1/5 的流量是跑在混部集群之上的。



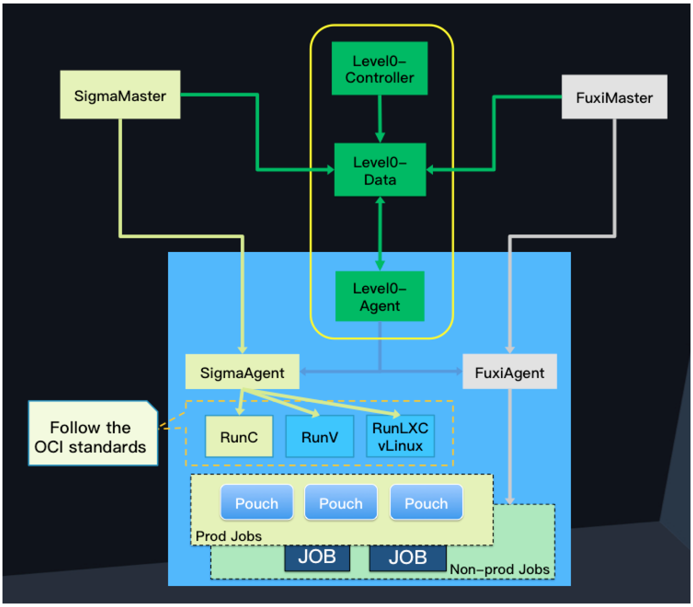
混部非混部集群资源使用对比图

在日常情况下，我们可以把在线服务的集群的 CPU 利用率从非混部的 10% 提升到混部的 40% 以上，整体的成本节省在 30% 以上。而在线受到的干扰在 5% 以内。



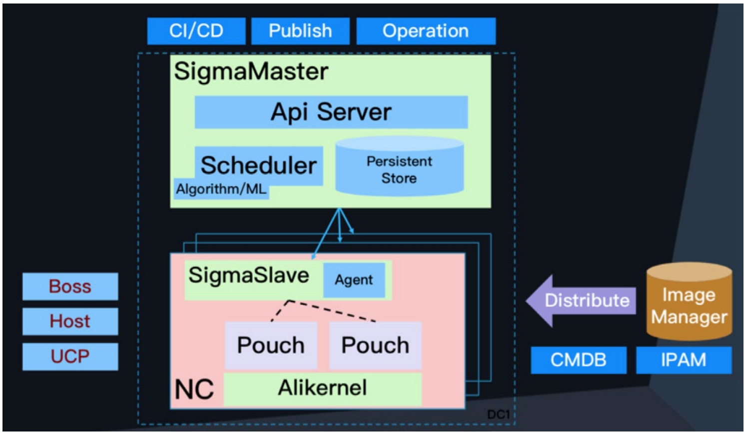
混部非混部集群平均服务响应时间对比图

混部调度的架构



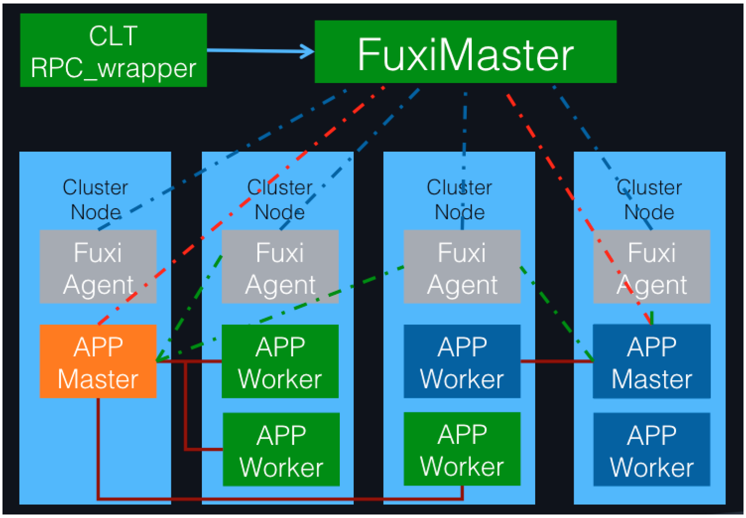
混部调度的架构示意图

在混部集群中，我们的两个调度平台同时自主运行，sigma 管理在线服务容器的调度，而 Fuxi 管理 ODPS 上的的计算任务。为了让这两个调度器能一起进行工作，在中间我们使用了零层与零层管控来协议两者之间的资源分配。



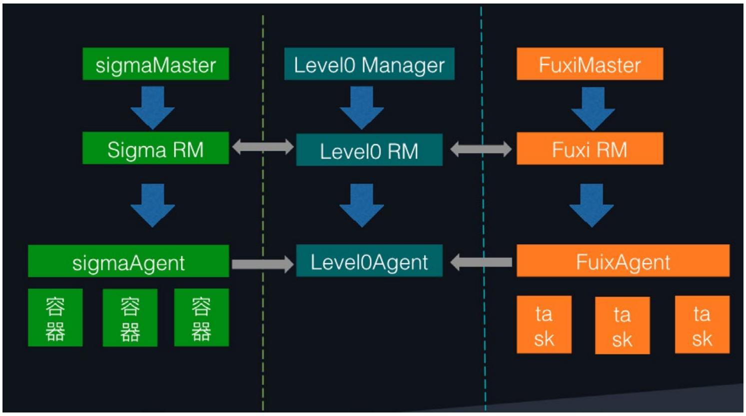
1. 在线服务容器调度器 Sigma 的特点是：

- 兼容 Kubernetes 的 API，和开源社区共建
- 采用阿里兼容 OCI 标准的 Pouch 容器
- 经历过阿里多年的大规模使用和双十一的验证



2. 计算任务调度器 Fuxi 的特点是：

- 而向海量数据处理和大规模计算类型的复杂应用
- 提供了一个数据驱动的多级流水线并行计算框架，在表述能力上兼容 MapReduce，Map-Reduce-Merge，Cascading，FlumeJava 等多种编程模式
- 高可扩展性，支持十万以上级的并行任务调度，能根据数据分布优化网络开销。自动检测故障和系统热点，重试失败任务，保证作业稳定可靠运行完成



3. 通过零层的资源协调机制，让整个集群平稳地管理并运行进来

- 混部集群管理
- 各调度租户之间的资源配比
- 日常与压测大促等时段的策略
- 异常检测与处理

混部的资源隔离

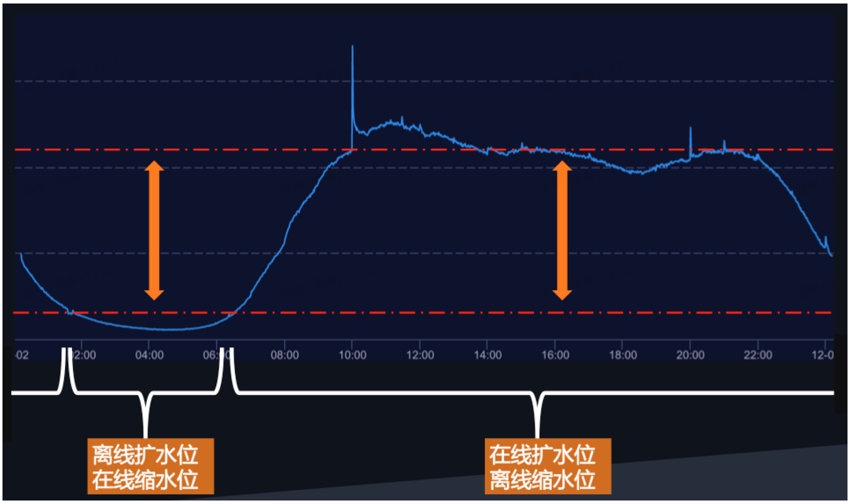
在混部中，放在首位的就是资源隔离问题，如果隔离问题没做好，竞争问题没解决，那就很容易引发线上的问题。轻一点，让用户的感官体验变差；重一点，引发线上故障，造成无法服务的恶劣影响。

而解决资源竞争的问题，主要从两个方面出发：

1. 调度：通过资源画像的技术，在资源真正发生竞争之前，就预测规划好，尽量减少这种情况发生的可能性。它是主动触发，可以不断优化，但延时较高。
2. 内核：在资源真正发生竞争的极端情况下，根据任务的优先级，如何做到既能保障高优先级任务不受影响，又能控制影响到的低优先级任务伤害最低。它是被动触发，保底的必须手段，生效快。

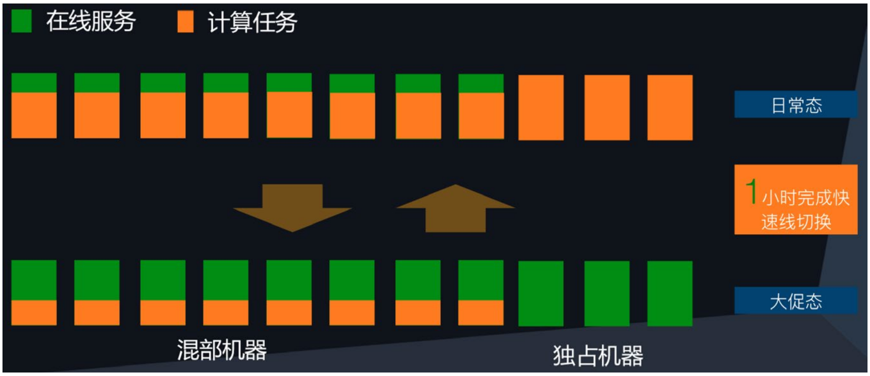
在调度上，我们主要从以下向个方面进行优化：

1. 日常的分时复用：由于波峰波谷的存在，在线服务与计算任务在一天中的峰值正好可以产生互补的情况，所以我们可以通过白天夜晚的分时复用提高资源的使用效率。



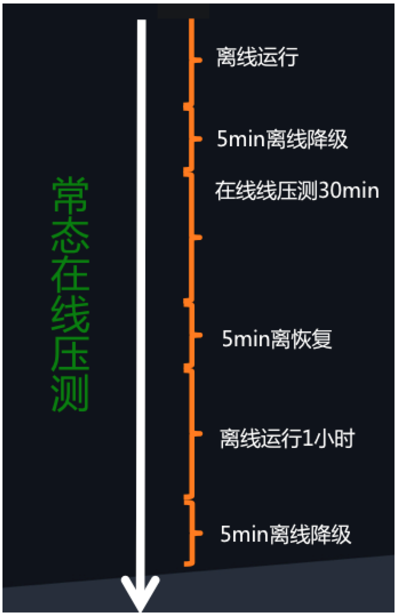
- 对集群进行资源使用的画像
- 在线服务凌晨 1–6 点为低峰，离线是高峰，针对这一特性进行水位调整
- 通过在线服务资源画像智能挑选空闲容器进行 offline 处理

2. 大促的分时复用：电商类业务由于大促的存在，在大促或压测的时候会产生比平时高几倍十几倍的压力差，如果这个时候对计算任务的资源进行降级让给在线服务全使用，就可以轻松地支撑起那短暂的脉冲压力。



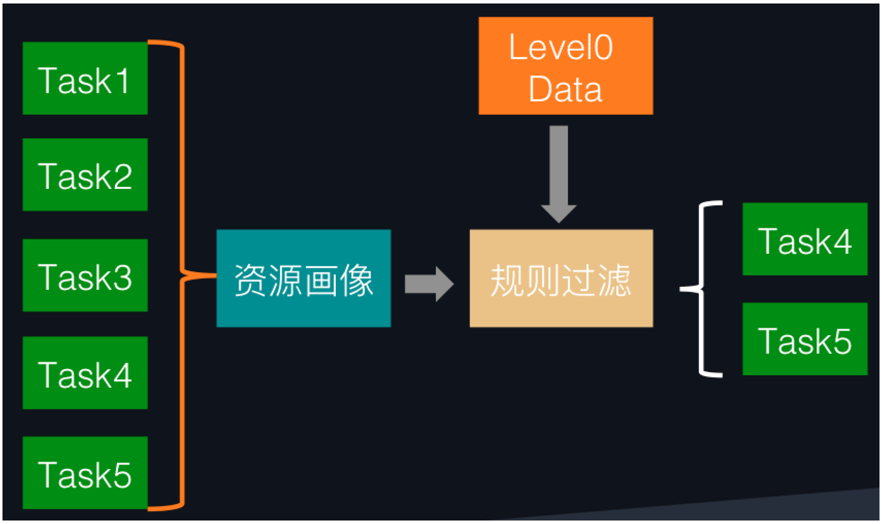
- 日常态，计算任务占用在线服务的资源
- 大促态，在线服务占用计算任务的资源
- 1 小时快速完成切换，提高资源的利用

3. 无损有损降级：在线服务会有一些特定的业务高峰时间，比如压测，比如大促等。那么如何在降级计算任务的时候，带来的影响尽可能小呢？这里我们就需要对降级的方案做特殊的处理。



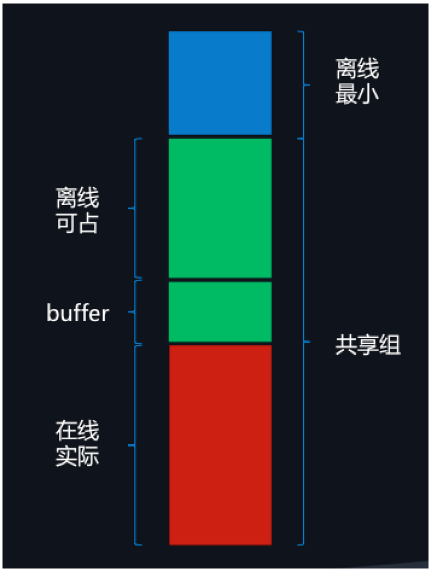
- 在无损降级：由于在线服务的 NC 平均利用率不高，再加上 70% 的计算任务小于 3 分钟，那么只要压测或大促在降级之后的 5 分钟，计算任务对于在线服务的干扰就不会那么大了。另一个问题是做到分钟级的恢复，这样只有当在线服务的真正高峰才会受到影响，而这个时间段又是比较短的，那么影响就会降低。
- 有损降级：当在线服务受到严重影响的时候，我们也可以做到秒级的 Kill，迅速恢复，让在线服务的影响降到最低。

4. 计算任务选取：计算任务我们比喻成沙子，但是沙子也有大有小，也需要对沙子进行筛选，才能把空隙填充满但又不溢出。



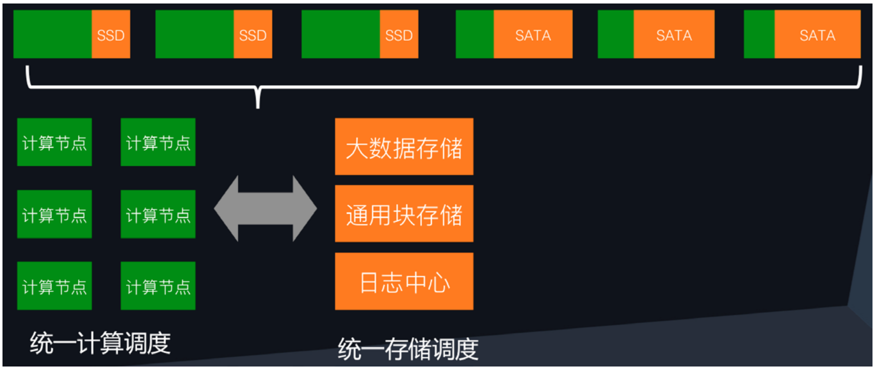
- 对作业进行资源使用的画像，分析出作业需要消耗的资源。
- 通过 0 层来获得宿主机剩余的确切计算资源能力
- 挑选符合条件的最佳作业，尽可能最大利用，也尽可能降低竞争。

5. 动态弹性内存：由于我们的存量资源并没有考虑到混部，内存与 CPU 都是按照在线服务原来的使用配比，并没有富余的内存存在，但是由于计算增加了，内存就成了瓶颈，原来在线服务以静态分配内存的方式就不再适合。



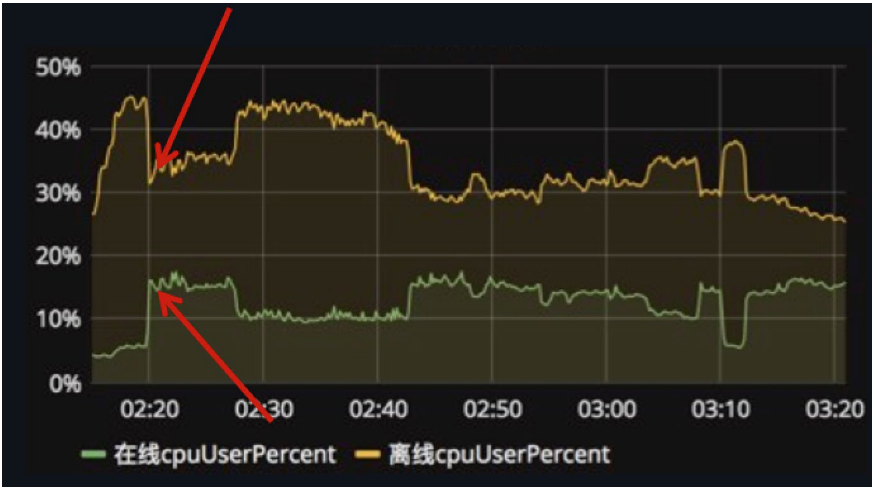
- 在线服务的内存加入共享分组
- 基于在线服务的实际内存使用，动态调整计算任务占用的内存水位
- 当在线服务压力突增时，通过迁移或 Kill 任务的方式，自动降级计算任务的内存水位。计算任务释放内存后，内核马上进行资源回收。
- 当整机发生 OOM 时，优先杀计算任务中优先级低的任务。

6. 存储计算分离：在线服务是重 IOPS，但是存储量不大，所以使用的都是 SSD 的小盘；而计算任务都是重存储量，但 IOPS 不大，所以使用的都是 HDD 的大盘。而在混部的时候，如果还是以本地盘的方式来处理数据，计算混杂在一起，对于调度复杂度是几何级的提升。所以我们需要把本地盘虚拟化成统一的存储池，通过远程的方式，根据不同的需求访问不一样的存储设备。另外阿里也开始大规模建设 25G 的网络设施，整个网络能力提升，也让远程访问变得像本地一样快。



内核隔离，我们主要从以下方面进行处理：

1. CPU 调度优化：这是隔离当中最重要的一项，当在线业务使用 CPU 压力上升，计算任务必须要毫秒级的自适性退出。



- CPU 抢占
  - 按照 CGroup 分配优先级 (cpu.shares)
  - 高优先级任务可以抢占低优先级任务的时间片
- 规避 HT（noise clean）
  - 避免离线任务调度到在线任务相邻的 HT 上
  - 保证已经运行的离线任务在在线任务于相邻 HT 上唤醒后迁走

- L3 Cache 隔离
  - 通过 BDW CPU 的特性 CAT 来进行对 Cache 访问的流量控制，进而达到限制低优先级计算任务对 CPU 的占用。
- 内存带宽隔离
  - Memory Bandwidth Monitoring ，通过时实监控来进行策略调整
  - Cfs bandwidth control 调节计算任务运行时间片长度。通过缩短时间片，让高优先级任务更容易获得占用 CPU 的机会。

2. 内存保护

- 内存回收隔离
  - 按照不同的 CGroup 分配优先级
  - 增加组内回收机制，避免全局内存回收干扰在线任务
  - 按优先级确定内存回收的权重，在线任务的内存被回收的更少
- OOM 优先级
  - 整机 OOM 时，优先杀低优先级任务

3. IO 分级限制

- 文件级别的 IO 带宽隔离 (上限)
  - 新增 blkio 的控制接口
  - 限制 IOPS，BPS
- 文件级别的保低带宽 (下限)
  - 允许应用超出保底带宽后使用富余的空闲带宽；
- Metadata throttle
  - 限制特定操作的 metadata 操作，例如一次性删除大量小文件。

4. 网络流量控制

- 带宽隔离
  - 隔离本机带宽（TC）。
  - Pouch 容器间的带宽隔离
- 带宽共享（金、银、铜）

- 在离线间可以存在共享带宽
- 进程间按照优先级可以抢占带宽

混部的未来规划

混部技术经过四年的磨炼，终于在 2017 支撑起了阿里巴巴双十一核心交易流量的 20%，并且也作为阿里巴巴未来数据中心建设的标准技术。而在未来的一年中，混部技术会向更精细化的调度能力演进。

在场景上，会更多元化，无论是实时计算，还是 GPU，甚至是 FPGA，都能够在混部在一起。在规模上，也会从十万核级别的集群往百万核级别的集群扩展。在资源画像能力上，会引入更多的深度学习，提高预测的准确性，为利用率再次大幅度提升打基础。在调度能力上，会建立更加完善的优先级体系，在资源分配与协调上不会以在线服务和计算任务来区别，而是以通用的优先级来调度，解决更多资源类型部混问题。总结一句话，让混部真正成为调度的一种通用能力。

经历 400 多天打磨，HSF 的架构和性能有哪些新突破？

思邪

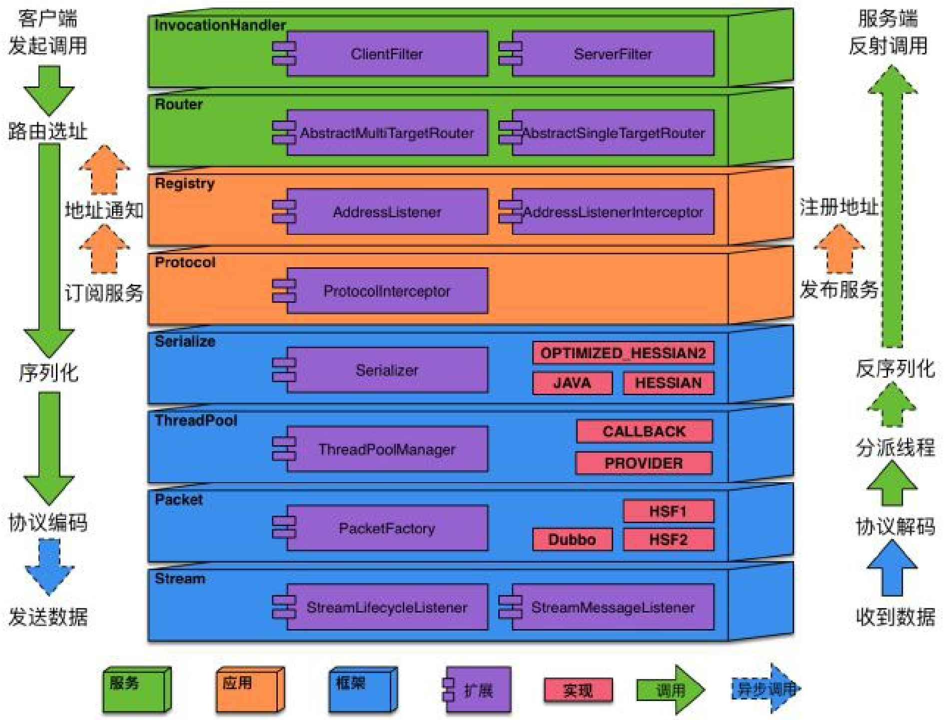
阿里妹导读：2017 年的双十一圆满结束了，1682 亿的成交额再一次刷新了记录，而 HSF（High Speed Framework，分布式服务框架）当天调用量也突破了 3.5 万亿次，调用量是 2016 年双十一的三倍多。为了这不平凡的一天中，HSF 经历了一年多的磨练，从架构升级到性能优化，从可用性提升到运维提升，完成了从 2.1 到 2.2 的全面升级，截止到双十一开始的一刻，HSF2.2 在全网机器占比超过 75%，并成功接受了大促考验。

从 HSF2.1 到 HSF2.2 的目标是提升 HSF 框架的扩展性、易用性和性能，相当于将原有 HSF 推倒了重新来过，因此涉及到的内容众多，这里不做展开，我们仅从架构升级、性能优化和运维提升三个方面来聊一下 HSF2.2 做了什么，有什么能帮到大家，以及如何解决了双十一的一些问题。

架构升级

HSF2.1 主要被用户抱怨的一点就是扩展性不强，比如：不少用户想拦截调用请求但无法方便做到，只好退而求其次，使用代理来包装 HSF 的客户端或者服务端，这样做一来显得繁琐，二来出现问题不易排查。不光是用户的扩展支持的不好，框架自身对不同的开源组件适配也显得力不从心，比如：支持 ZooKeeper 作为注册中心的方案一直难以实施，原因在于框架内部与 ConfigServer 耦合过于紧密。无论是用户还是框架的维护者，都受限于现有架构，这是 HSF 需要改变的地方。

HSF2.2 的架构与原有版本有了显著变化，首先是层与层之间的边界清晰，其次组件之间的关系职责明确，最后是扩展可替换的思想贯穿其中，下面我们先看一下新的架构：



可以看到 HSF2.2 从宏观上分为了三个域：框架、应用和服务，其中框架是对上层多个应用做支撑的基础，而服务属于具体的某个应用，从这种划分可以看出，

HSF2.2 是天然支持多应用（或多模块）部署的。每个域中，都有各自的功能组件，比如：服务域中的 InvocationHandler 就是对服务调用逻辑的抽象，通过扩展其中的 ClientFilter 或者 ServerFilter 完成对客户端或者服务端的调用处理工作，这里只是罗列了主要的功能组件和扩展点，以下是对它们的说明：

服务，对应于一个接口，它可以提供服务或者消费服务

组件名	扩展点	说明
InvocationHandler	ClientFilter	客户端调用拦截器，扩展它实现调用端的请求拦截处理
InvocationHandler	ServerFilter	服务端调用拦截器，扩展它实现服务端的请求拦截处理
Router	AbstractMultiTargetRouter	客户端调用多节点选择路由器，扩展它实现客户端调用多节点的路由
Router	AbstractSingleTargetRouter	客户端调用单节点选择路由器，扩展它实现客户端调用单节点的路由

应用，对应于一个应用或者一个模块，它管理着一组消费或者发布的服务

组件名	扩展点	说明
Registry	AddressListener	注册中心地址监听器，扩展它可以获取到注册中心推送的地址列表
Registry	AddressListenerInteceptor	注册中心地址拦截器，扩展它可以监听到注册中心推送地址并加以控制
Protocol	ProtocolInterceptor	订阅发布流程，扩展它可以捕获应用发布或者消费的具体服务

框架，对应于框架底层支持，它统一管理着线程、链接等资源

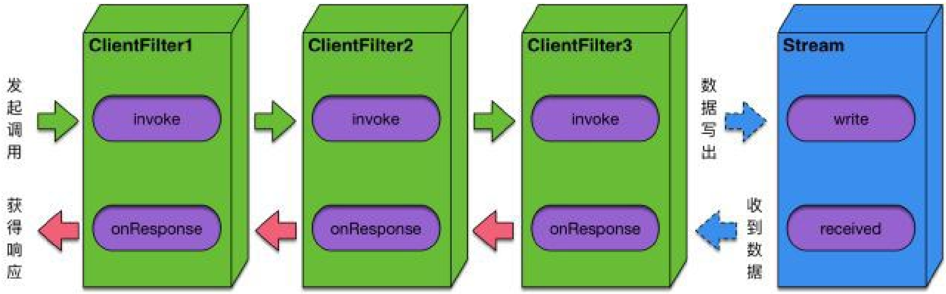
组件名	扩展点	说明
Serialize	Serializer	序列化和反序列化器，需要实现新的序列化方式可以扩展它
Packet	PacketFactory	协议工厂，适配不同的RPC框架协议需要扩展它
Stream	StreamLifecycleListener	长连接生命周期管理监听器，扩展它可以监听到长连接的建立和销毁等生命周期
Stream	StreamMessageListener	长连接事件监听器，扩展它可以监听到每条连接上跑的数据

分类介绍了 HSF2.2 的主要组件后，我们来看看架构的改变，扩展性的提升，能够为业务方，为他们更好的支持双十一带来什么。

纯异步调用拦截器

HSF 异步 future 调用被业务方越来越多的使用，但是 eagleeye 对 future 调用的耗时统计上一直不准，因为 future 调用返回的是默认值，所以业务看到的是一次飞快的调用返回，它真实的耗时该如何度量？信息平台承载了集团的众多内部服务业务，一直苦于没有办法全面的监控 HSF 的调用记录，也无法做到异常报警，难道还要让所有的使用方配置代理？

HSF2.2 设计的 InvocationHandler 组件以及扩展，完美的解决了这个问题，我们近距离观察一下 InvocationHandler 以及其扩展，以客户端调用场景为例：一次调用是如何穿过用户扩展的调用拦截器的。



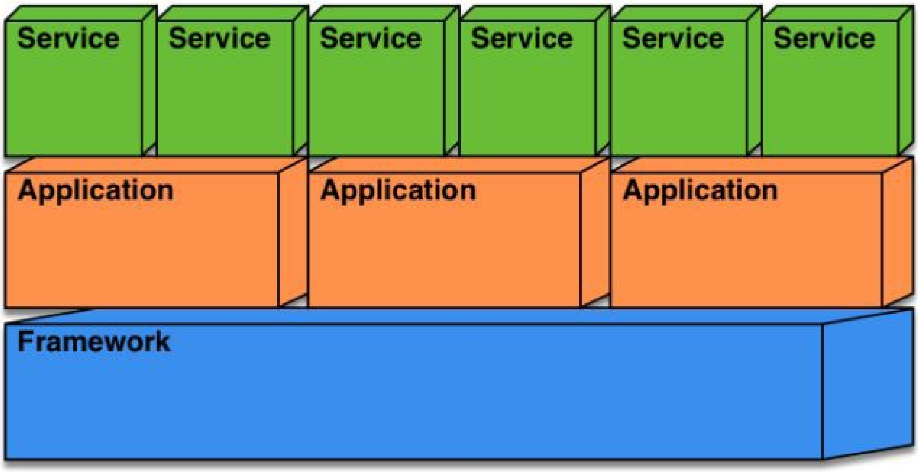
可以看到，HSF2.2 定义的调用拦截器不是一个简单的正向 invoke 过程，它还具备响应回来时的逆向 onResponse 过程，这样一来一回很好的诠释了 HSF 同步调用异步执行的调用模型。用户可以通过在一次调用中计算 invoke 和 onResponse 之间的时差来获得调用的正确耗时，而不用关心 HSF 调用是同步还是异步，用户也可以扩展一个调用拦截器穿插在其中，用于监控 HSF 的调用内容，而不用让应用方做任何代码上的改造。

多应用支持

多模块部署一直是业务方同学探索的热点，从合并部署开始，HSF 就有限的支持了多应用部署，但是一直没有将其融入到架构中。在 HSF2.2 中，应用不仅仅是作为托管服务的容器而是成为核心领域，多应用序列化的难题，就在应用层解决。

（模块在 HSF2.2 中也会被认为是一个应用，以下提到模块或者应用可以认为是一个概念。）

HSF2.2 支持多个应用部署的结构如下图所示：



可以看到，HSF2.2 框架域处于最底层，它向上为应用提供了线程、连接等多种资源的支持，而应用能够发布或者订阅多个服务，这些服务能够被应用统一管理，也支持更加精细化的配置，层与层的司职明确让 HSF 具有了良好的适应性。

性能优化

RPC 流程中，序列化是开销很大的环节。尤其现在业务对象都很复杂，序列化开销更不容小觑。

下面先介绍下今年 HSF 针对 Hessian 的几个优化点。也感谢分享思路和反馈问题的坤谷、道延和昕辰三位同学。

优化 1：元数据共享

hessian 序列化会将两种信息写到输出流：

1. 元数据：即类全名，字段名
2. 值数据：即各个字段对应值（如果字段是复杂类型，则会递归传递该复杂类型的元数据和内部字段的值数据）

在 hessian1 协议里，每次写出 Class A 的实例时，都会写出 Class A 的元数据和值数据，就是会重复传输相同的元数据。针对这点，hessian2 协议做了一个优化就是：在“同一次序列化上下文”里，如果存在 Class A 的多个实例，只会对 Class A 的元数据传输一次。该元数据会在对端被缓存起来重复使用，下次再序列化 Class A 的对象时，只需要先写出对元数据的一个引用句柄（缓存中的 index，用一个 int 表示），然后直接写出值数据即可。接受方通过元数据句柄即可知道后面的值数据对应的类型。

这是一个极大的提升。因为编码字段名字（就是字符串）所需的字节数很可能比它对应的值（可能只是一个 byte）更多。

不过在官方的 hessian 里，这个优化有两个限制：

1. 序列化过程中类型对应的 Class 结构不能改变
2. 元数据引用只能在“同一个序列化上下文”，这里的“上下文”就是指同一个 HessianOutput 和 HessianInput。因为元数据的 id 分配和缓存分别是在 HessianOutput 和 HessianInput 里进行的

限制 1 我们可以接受，一般 DO 不会再运行时改变。但是限制 2 不太友好，因

为针对每次请求的序列化和反序列化，HSF 都需要使用全新构造的 HessianOutput 和 HessianInput。这就导致每次请求都需要重新发送上次请求已经发送过的元数据。

针对限制 2，HSF 实现了跨请求元数据共享，这样只要发送过一次元数据，以后就再也不用发送了，进一步减少传输的数据量。实现机制如下：

1. 修改 hessian 代码，将元数据 id 分配和缓存的数据结构从 HessianOutput 和 HessianInput 剥离出来。
2. 修改 HSF 代码，将上述剥离出来的数据结构作为连接级别的上下文保存起来。
3. 每次构造 HessianOutput 和 HessianInput 时将其作为参数传入。这样就达了跨请求复用元数据的目的。

该优化的效果取决于业务对象中，元数据所占的比例，如果“精心”构造对象，使得元数据所占比例很大，那么测试表现会很好，不过这没有意义。我们还是选取线上核心应用的真实业务对象来测试。从线上 tcp dump 了一个真实业务对象，测试同学以此编写测试用例得到测试数据如下：

1. 新版本比老版本 CPU 利用率下降 10% 左右
2. 新版本的网络流量相比老版本减少约 17%

线上核心应用压测结果显示数据流量下降一般在 15%~20% 之间。

优化 2：UTF8 解码优化

hessian 传输的字符串都是 utf8 编码的，反序列化时需要进行解码。hessian 现行的解码方式是逐个字符进行。代码如下：

```
private int parseUTF8Char() throws IOException {   int ch = _offset < _length
? (_buffer[_offset++] & 0xff) : read();
if (ch < 0x80)
    return ch;
  else if ((ch & 0xe0) == 0xc0) {
   int ch1 = read();
    int v = ((ch & 0x1f) << 6) + (ch1 & 0x3f);        return v;
  } else if ((ch & 0xf0) == 0xe0) {
     int ch1 = read();
```

```
  int ch2 = read();
   int v = ((ch & 0x0f) << 12) + ((ch1 & 0x3f) << 6) + (ch2 & 0x3f);
    return v;
 } else
  throw error("bad  utf-8  encoding  at  " + codeName(ch));
}
```

UTF8 是变长编码，有三种格式：

1 byte format: 0xxxxxxx

2 byte format: 110xxxxx 10xxxxxx

3 byte format: 1110xxxx 10xxxxxx 10xxxxxx

上面的代码是对每个字节，通过位运算判断属于哪一种格式，然后分别解析。

优化方式是：通过 unsafe 将 8 个字节作为一个 long 读取，然后通过一次位运算判断这 8 个字节是否都是“1 byte format”，如果是（很大概率是，因为常用的 ASCII 都是“1 byte format”），则可以将 8 个字节直接解码返回。以前 8 次位运算，现在只需要一次了。如果判断失败，则按老的方式，逐个字节进行解码。主要代码如下：

```
private boolean parseUTF8Char_improved() throws IOException {
 while (_chunkLength > 0) {
    if (_offset >= _length && !readBuffer()) {
      return false;
   }
   int sizeOfBufferedBytes = _length - _offset;        int toRead =
sizeOfBufferedBytes <= _chunkLength ? sizeOfBufferedBytes : _chunkLength;
   // fast path for ASCII
    int numLongs = toRead >> 3;
   for (int i = 0; i < numLongs; i++) {
       long currentOffset = baseOffset + _offset;            long test =
unsafe.getLong(_buffer, currentOffset);
    if ((test & 0x8080808080808080L) == 0L) {               _chunkLength -=
8;
       toRead -= 8;
       for (int j = 0; j < 8; j++) {                     _sbuf.append((char)(_
buffer[_offset++]));                 }
    } else {
      break;
       }
```

```
  }
    for (int i = 0; i < toRead; i++) {            _chunkLength--;
   int ch = (_buffer[_offset++] & 0xff);
       if (ch < 0x80) {                 _sbuf.append((char)ch);
         } else if ((ch & 0xe0) == 0xc0) {
       int ch1 = read();
     int v = ((ch & 0x1f) << 6) + (ch1 & 0x3f);

           _sbuf.append((char)v);
        } else if ((ch & 0xf0) == 0xe0) {
             int ch1 = read();
        int ch2 = read();
       int v = ((ch & 0x0f) << 12) + ((ch1 & 0x3f) << 6) + (ch2 & 0x3f);

           _sbuf.append((char)v);
        } else
        throw error("bad utf-8 encoding at " + codeName(ch));
    }
   }
  return true;
}
```

同样使用线上 dump 的业务对象进行对比，测试结果显示该优化带来了 17% 的性能提升：

time: 981

improved utf8 decode time: 810

(981-810)/981 = 0.1743119266055046

优化 3：异常流程改进

hessian 在一些异常场景下，虽然业务不一定报错，但是性能却很差。今年我们针对两个异常场景做了优化。

UnmodifiableSet 构造异常

某核心应用压测发现大量如下错误：

```
java.util.Collections$UnmodifiableSet
============================================================================
java.lang.Throwable.<init>(Throwable.java:264)
java.lang.Exception.<init>(Exception.java:66)
```

```
java.lang.ReflectiveOperationException.<init>(ReflectiveOperationException.
java:56)
java.lang.InstantiationException.<init>(InstantiationException.java:63)
java.lang.Class.newInstance(Class.java:427)
com.taobao.hsf.com.caucho.hessian.io.CollectionDeserializer.
createList(CollectionDeserializer.java:107)
com.taobao.hsf.com.caucho.hessian.io.CollectionDeserializer.
readLengthList(CollectionDeserializer.java:88)
com.taobao.hsf.com.caucho.hessian.io.Hessian2Input.readObject(Hessian2Input.
java:1731)
com.taobao.hsf.com.caucho.hessian.
io.UnsafeDeserializer$ObjectFieldDeserializer.deserialize(UnsafeDeserializer.
java:387)
```

这是 Hessian 对 UnmodifiableSet 反序列化的一个问题，因为 Unmodifiable-Set 没有默认构造函数，所以 Class.newInstance 会抛出，出错之后 Hessian 会使用 HashSet 进行反序列化，所以业务不会报错，但是应用同学反馈每次都处理异常，对性能影响比较大。

另外 Collections$UnmodifiableCollection，Collections$UnmodifiableList，Collections$UnmodifiableMap，Arrays$ArrayList 也有一样的问题。

针对这些构造函数一定会出错的类型，我们修改了 Hessian 的代码直接跳过构造函数的逻辑，较少不必要的开销。

重复加载缺失类型

某核心应用压测发现大量线程 block 在 ClassLoader.loadClass 方法

```
"HSFBizProcessor-DEFAULT-7-thread-1107" #3049 daemon prio=10 os_
prio=0 tid=0x00007fd127cad000 nid=0xc29 waiting for monitor entry
[0x00007fd0da2c9000]
java.lang.Thread.State: BLOCKED (on object monitor)
at com.taobao.pandora.boot.loader.LaunchedURLClassLoader.
loadClass(LaunchedURLClassLoader.java:133)
- waiting to lock <0x0000000741ec9870> (a java.lang.Object)
atjava.lang.ClassLoader.loadClass(ClassLoader.java:380)
at java.lang.Class.forName0(Native Method)
at java.lang.Class.forName(Class.java:348)
at com.taobao.hsf.com.caucho.hessian.io.SerializerFactory.
getDeserializer(SerializerFactory.java:681)
```

排查确认是业务二方包升级，服务端客户端二方包版本不一致，导致老版本的一

边反复尝试加载对应的新增类型。虽然该问题并没有导致业务错误，但是确严重影响了性能，按应用同学的反馈，如果每次都尝试加载缺失类型，性能会下降一倍。

针对这种情况，修改 hessian 代码，将确认加载不到的类型缓存起来，以后就不再尝试加载了。

优化 4：map 操作数组化

大型系统里多个模块间经常通过 Map 来交互信息，互相只需要耦合 String 类型的 key。常见代码如下：

```
public static final String key = "mykey";
Map<String,Object> attributeMap = new HashMap<String,Object>();
Object value = attributeMap.get(key);
```

大量的 Map 操作也是性能的一大消耗点。HSF 今年尝试将 Map 操作进行了优化，改进为数组操作，避免了 Map 操作消耗。新的范例代码如下：

```
public static final AttributeNamespace ns = AttributeNamespace.
createNamespace("mynamespace");
public static final AttributeKey key = new AttributeKey(ns, "mykey");
DefaultAttributeMap attributeMap = new DefaultAttributeMap(ns, 8);
Object value = attributeMap.get(key);
```

工作机制简单说明如下：

1. key 类型由 String 改为自定义的 AttributeKey，AttributeKey 会在初始化阶段就去 AttributeNamespace 申请一个固定 id
2. map 类型由 HashMap 改为自定义的 DefaultAttributeMap，DefaultAttributeMap 内部使用数组存放数据
3. 操作 DefaultAttributeMap 直接使用 AttributeKey 里存放的 id 作为 index 访问数组即可，避免了 hash 计算等一系列操作。核心就是将之前的字符串 key 和一个固定的 id 对应起来，作为访问数组的 index

对比 HashMap 和 DefaultAttributeMap，性能提升约 30%。

HashMap put time(ms) : 262

ArrayMap put time(ms) : 185

HashMap get time(ms) : 184

ArrayMap get time(ms) : 126

在刚刚过去的 2017 年双 11 中，HSF 服务治理（HSFOPS）上线了诸多提升应用安全、运维效率的新功能，如：

1. 服务鉴权
2. 单机运维
3. 连接预热

其中，在今年双 11 中发挥最大价值的，当属 HSF 连接预热功能。

HSF 连接预热功能，可以帮助应用提前建立起和 Providers 之间的连接，从而避免首次调用时因连接建立造成的 rt 损耗。

HSF 服务治理平台的连接预热功能，支持指定应用、机器分组以及单元环境的预热，并且可以指定预热执行计划，并支持按预热批次暂停、恢复预热。在 2017 双 11 预热完成后，目标应用集合整体连接数增长大于 50%，部分应用增长达 40 倍，符合连接预热的预期。

小结

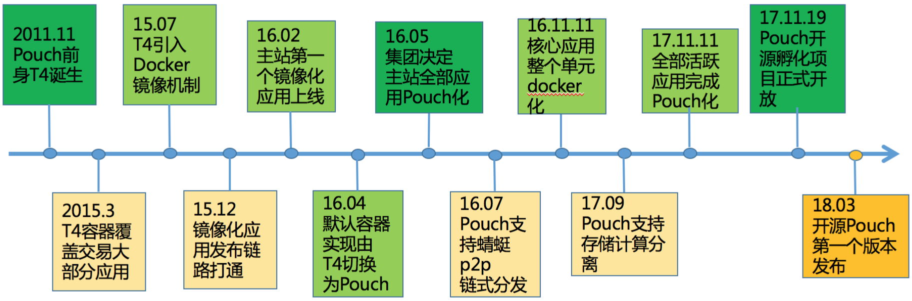
架构升级使 HSF 能够从容的应对未来的挑战，性能优化让我们在调用上追求极致，运维提升更是将眼光拔高到全局去思考问题，这些改变共同的组成了 HSF2.1 到 HSF2.2 的变革，而这次变革不仅仅是稳定的支撑 2017 年的双十一，而是开启了服务框架下一个十年。

直击阿里容器技术 Pouch

沈陵

阿里集团内部使用的 Pouch 容器技术源于 2011 年开始开发的 t4 容器，融合了社区主流容器技术轻量、自洽、一次打包多次运行的优点，专注于稳定性、性能和隔离性方面的优化，适配了阿里内部的使用场景和开发的使用习惯，是阿里集团内部容器化、镜像化、混部和统一调度的基石。使用场景已经覆盖了集团大部分的 BU 的上万个应用，集团内部在线业务 100% 都做 Pouch 化。

Pouch 的技术演化过程

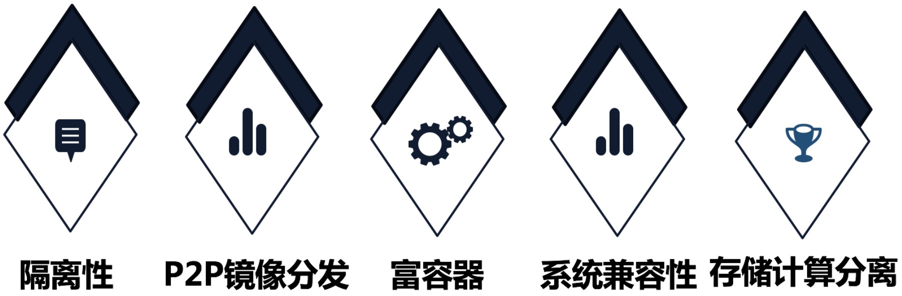


2011 年以前集团主要是使用物理机或基于 xen 的虚拟机。随着应用从大到小的拆分以及对提高资源利用率的追求，物理机或虚拟机不再能满足我们的要求，Pouch 的雏型 t4 应运而生，到 2015 年中交易大部分应用都已经切换到 Pouch 上面，这时候开源的容器技术 Docker 在开发者社区已经很火热了，很多开发都在本地构建 docker 镜像来模拟运行时环境，我们调研了 docker 的技术栈和生态后发现 Docker 镜像技术正好能弥补 Pouch 在环境定义方面的不足，我们决定把 Docker 的镜像功

能加入 Pouch，这样开发人员在本地构建的镜像就可以直接在线上用 Pouch 运行起来，保证了开发测试正式环境的一致性，运行时继续使用 Pouch 原来开发的 lxc 技术。后来随着 oci 运行时规范的成熟我们把 lxc 包装成了符合 oci 运行时规范的一个 runtime 实现。当然镜像技术也不是完美的，当全部应用都从发主包变成发镜像后，随之来而的一个问题就是应用扩容或发布的时候需要下载的包大小从很小的主包，变成了一个相对比较大的镜像，这样在批量的发布或扩容的时候镜像仓库的出口网络会成为瓶颈，扩容或发布的速度也变得很慢，并且影响成功率。为了解决这个问题 Pouch 在 2016 年中支持了使用蜻蜓做链式的镜像分发，不需要所有的内容都从中心的镜像仓库下载。到 2017 年双 11 集团内部的所有活跃应用都已经 Pouch 化，为了把这么多年容器方面的经验和技术回馈给社区，我们已经开始做 Pouch 的开源孵化，预计到 2018.3 月底 Pouch 的第一个正式的开源版本发布。

Pouch 如何支持双 11 大促

电商双 11 大促的场景对容器技术的考验主要在于几个方面：不同容器之间的隔离性、离线进程和在线容器之间的隔离性都需要保证。大促期间很多应用都有上万个实例，这些应用同时发布时拉取镜像的成功率和效率的保证也是对容器技术的一大难题。为了和内部相关的系统集成容器中不能只启动用户的一个进程，大促相关的流量调度、监控、日志以及与安全体系的集成都需要在用户的容器中启动一些辅助服务，这就是 Pouch 支持的富容器技术的需求来源。同时大促期间我们能用到的宿主机也是各式各样的，比如有长尾应用一直在用的物理机，有新采购的物理机，也有从阿里云买到的 ECS，这些物理机上面的系统和内核版本也都不尽相同，这样就要求 Pouch 要支持所有可用的系统版本和内核版本，内核版本从 2.6.32 到 3.10，alios 版本从 5u 到 7u 都需要支持。大规模的使用场景中经常会有物理机挂掉，为了不影响服务，一个容器挂了后会调度编排系统会立即再扩容一个做为补充，这种方式对于无状态的应用是适用的，但是如果应用在磁盘上面存储了必要的数据就没办法这么替换了，Pouch 配合盘古分布式块存储支持申请存储和计算分离的容器，这样即使物理挂掉了，这个容器在另一个物理机上面拉起后还能继续访问以前保存的数据。



隔离性

应用容器化以后每个容器可以使用的资源都是由调度系统分配的，不再是整个宿主机的资源。Cpu 和内存可以通过 cgroup 做隔离，但是如果容器通过 top、free 等指令看到的还是整个宿主机的还是容易引起误解，还有像 nginx、nodejs、golang 等写的一些应用在启动阶段就会根据 cpu 个数来决定启动多少个进程，这就要求我们必须要做这方面的隔离，包含对磁盘使用量的隔离。用户态的方案像 lxcfs 这些的性能比较差，我们的方案是这三方面的用户视图隔离放在 alikernel 中实现。用户在容器能看到 top、free、df 等指令，包括看到 /proc 文件系统都是隔离过的，这样监控系统也可以直接运行在容器中来收集数据。

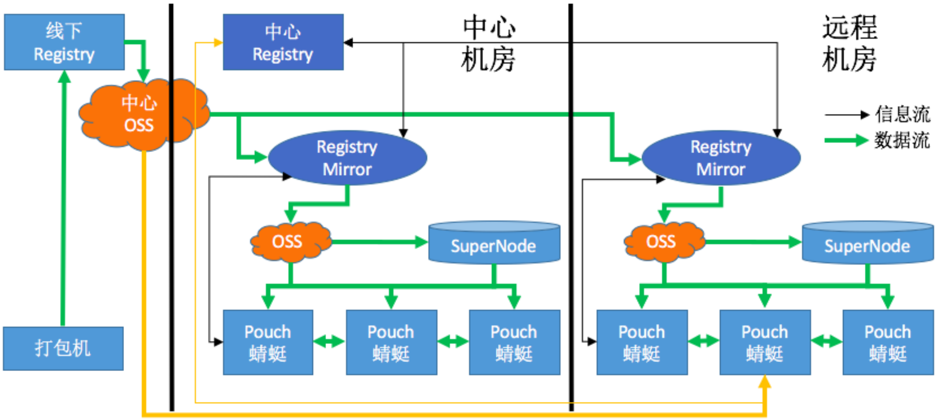
nproc、nofile 这部分的隔离由于低版本内核对这两种资源的使用量是记录在用户 uid 上面的，所以我们给每个容器分配了一个 uid 不同的 admin 用户，这样每个容器对文件和进程的使用量也都是隔离开的。

当离线和在线混部在一起的时候，上面说的这些隔离特性就不够用了，配合 alikernel 的最新版本 Pouch 还支持网络带宽隔离、磁盘 buffer r/w bps 和 meta tps 的隔离、cpu 三缓存隔离、cpu 调度优先级隔离、noisy clean 以及单 cgroup 内存回收水位控制等细粒度的隔离。

支持 p2p 镜像分发

我们 2011 年上线 t4 的时候和 2013 年 docker 推出的容器技术最大的不同就是我们当时没有用镜像来记录运行时环境，我们内部使用的是基线 -- 一个用来记录各个应用运行时依赖的系统。正是因为有了镜像这种创新才使得 build ship run 的理念

才为现实，所以 Pouch 也借鉴了镜像的概念。



镜像化之前我们只需要分发应用的代码编译出来的主包，现在是需要分发整个镜像。借助于镜像分层的特性，如果运行环境不经常变化，好的 Dockerfile 规范可以让镜像分发达到和分发主包一样的效果。但是依赖环境的变化会让这种情况恶化，特别是很多应用的部署并发度上千，并且会部署在不同的机房、不同的地域、不同的大陆。为了提升镜像下载的效率，Pouch 支持了多种 p2p 镜像加速插件，包括蜻蜓、minion 等。不同的机房都搭建了超节节点来加速机房之间的传输。对于有盘古的机房 Pouch 还支持使用盘古远程盘来做镜像，这样任何一个镜像每个机房都只需要下载一次，所有使用到这个镜像的机器都是挂载同一份只读数据盘。

镜像化的另一个特点是每次应用发布都需要重启容器，对于一些静态资源的发布不太友好，Pouch 还支持了对镜像的热更新，热更新的时候会比较两个镜像之间是不是只更新了指定的静态资源，否则就还是全量更新。这样就在保持镜像作为应用版本的唯一标识的情况下支持了运行时的 patch 更新。

富容器模式

容器社区提倡的一个容器一个进程的使用方式，对于微服务或新的应用比较合适，但是对于有上万个应用需要迁移到容器来还强制这种方式就有点无法实现了，为了让所有应用都能无缝的切换过来，我们还是在容器中用 /sbin/init 来拉起运维依赖的系统服务，最后切换到 admin 用户来拉起用户指定的 Entrypoint。这样做的好处

不只是照顾用户原来的习惯，用户不再需要自己管理进程，和容器退出信号这些事情。用户只需要指定了 Entrypoint 启动阶段就会拉起服务，需要系统做健康检查和优雅下线的，也只需要在特定位置放置 health 和 stop 脚本就可以了。

富容器还有一些好处是运维流程可以在应用的 Entrypoint 启动之前做一些事情，比如统一要做一些安全相关的事情，运维相关的 agent 拉起。这些需要统一做的事情放到用户的启动脚本，或镜像中就不合适了，富容器可以透明的处理掉这些事情。

系统和内核兼容性

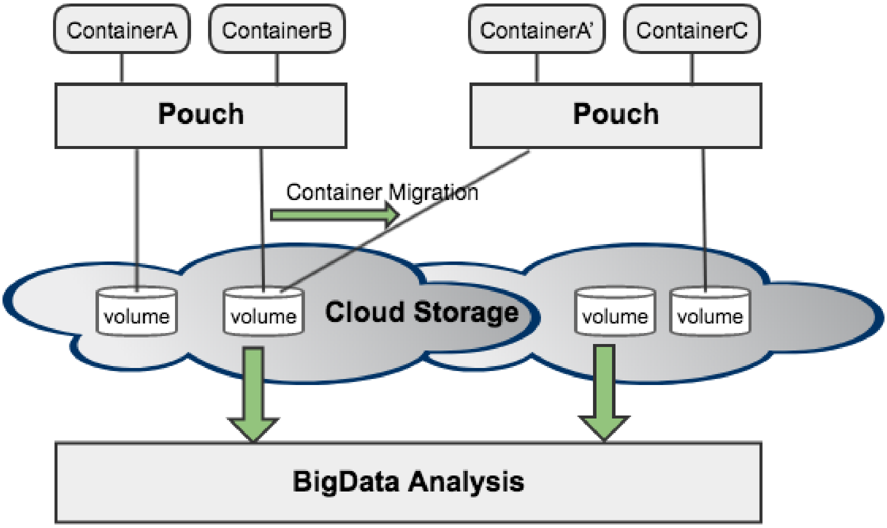
阿里巴巴集团内部都是使用 alios，其中的内核版本从 2.6.32 到 3.10 都有，要想让所有应用都运行在 Pouch 容器中，Pouch 就必须支持所有这些内核版本。好在是最低的 2.6.32 版本对 namespace 的支持也只是缺少 user namespace，其它的 namespace 包括常用的 cgroup 子系也都支持。但是像 /proc/self/ns 这些用来记录 namespace 的辅助文件是不存在的，setns 这些 ns 相关的系统调用也只有高版本内核才有。涉及到 ns 操作的部分在低版本内核上就需要用各种方法来做绕过，比如用 pid 来记录另一个 ns，而不是用引用了那一个 ns 的 fd。

为了适配各种网络模式，我们还引入了对 libnetwork 的支持，libnetwork 有在根 ns 中对子 ns 的网络相关的 netlink 系统调用，这个场景就是一个在低版本内核上面的难点，解决了这个问题后所有支持 libnetwork 的网络插件都可以在 Pouch 上面直接使用。我们还把 t4 使用的 bridge、vlan 和弹内云上使用的 vpc overlay 网络都做成了 libnetwork 的网络插件，并对性能要求比较高的场景支持了 sirvo 的网络插件。

关于分层文件系统，2011 年的时候 alikernel 就把 overlayfs backport 回来了，所以我们现在用的最低版本的内核也是有分层文件系统支持的，我们原来用的 t4 容器就是用这个特性来做镜像文件和它的 page cache 的共享。

除了向下对内核版本的适配，向上还有对各种运维系统、监控系统、用户使用习惯的适配和对接，最无缝的方式就是采用富容器模式。

支持计算存储分离



Pouch 实现了将容器的计算和存储资源进行分离，利用共享存储的作为容器存储，可实现有状态的应用容器的快速迁移，可实现无状态应用的日志数据快速的做离线大数据分析。利用通用的 volume 存储管理模块，可快速实现包括本地数据盘，内存盘，分布式存储的远程盘（阿里云盘古分布式存储）在容器中使用的切换能力，配合存储管理的调度系统，能够更灵活的利用存储资源。能有效的解决服务器存储资源不足或者浪费的场景问题。

以上就是我们 Pouch 容器在阿里内部的演进过程以及对双 11 这种大规模使用场景下所做的特殊支持，既支持现下最流行的容器使用方法，又兼顾使用习惯、隔离稳定性、性能这些方面的场景。

直击阿里双 11 神秘技术：PB 级大规模文件分发系统“蜻蜓”

如柏

阿里妹导读：2017 天猫双 11，交易峰值 32.5 万 / 秒，支付峰值 25.6 万 / 秒，数据库处理峰值 4200 万次 / 秒，再次刷新了记录。阿里集团基础设施蜻蜓，在双 11 期间，对上万台服务器同时下发 5GB 的数据文件，让大规模文件分发靠蜻蜓系统完美实现。

蜻蜓，通过解决大规模文件下载以及跨网络隔离等场景下各种难题，大幅提高数据预热、大规模容器镜像分发等业务能力。月均分发次数突破 20 亿次，分发数据量 3.4PB。其中容器镜像分发比 natvie 方式提速可高达 57 倍，registry 网络出口流量降低 99.5% 以上。今天，阿里妹邀请阿里基础架构事业群高级技术专家如柏，为我们详述蜻蜓从文件分发到镜像传输的技术之路。

蜻蜓的诞生

随着阿里业务爆炸式增长，2015 年时发布系统日均的发布量突破两万，很多应

用的规模开始破万，发布失败率开始增高，而根本原因就是发布过程需要大量的文件拉取，文件服务器扛不住大量的请求，当然很容易想到服务器扩容，可是扩容后又发现后端存储成为瓶颈。此外，大量来自不同 IDC 的客户端请求消耗了巨大的网络带宽，造成网络拥堵。

同时，很多业务走向国际化，大量的应用部署在海外，海外服务器下载要回源国内，浪费了大量的国际带宽，而且还很慢；如果传输大文件，网络环境差，失败的话又得重来一遍，效率极低。

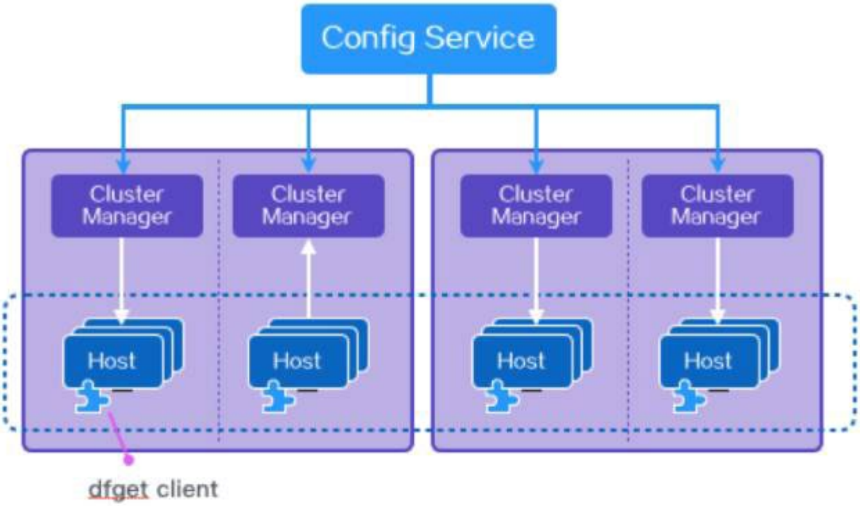
于是很自然的就想到了 P2P 技术，因为 P2P 技术并不新鲜，当时也调研了很多国内外的系统，但是调研的结论是这些系统的规模和稳定性都无法达到我们的期望。所以就有了蜻蜓这个产品。

设计目标

针对这些痛点，蜻蜓在设计之初定了几个目标：

1. 解决文件源被打爆的问题，在 Host 之间组 P2P 网，缓解文件服务器压力，节约跨 IDC 之间的网络带宽资源。
2. 加速文件分发速度，并且保证上万服务器同时下载，跟一台服务器下载没有太大的波动。
3. 解决跨国下载加速和带宽节约。
4. 解决大文件下载问题，同时必须要支持断点续传。
5. Host 上的磁盘 IO，网络 IO 必须可以被控制，以避免对业务造成影响。

系统架构



蜻蜓整体架构

蜻蜓整体架构分三层：第一层是 Config Service，他管理所有的 Cluster Manager，Cluster Manager 又管理所有的 Host，Host 就是终端，dfget 就是类似 wget 的一个客户端程序。

Config Service 主要负责 Cluster Manager 的管理、客户端节点路由、系统配置管理以及预热服务等等。简单的说，就是负责告诉 Host，离他最近的一组 Cluster Manager 的地址列表，并定期维护和更新这份列表，使 Host 总能找到离他最近的 Cluster Manager。

Cluster Manager 主要的职责有两个：

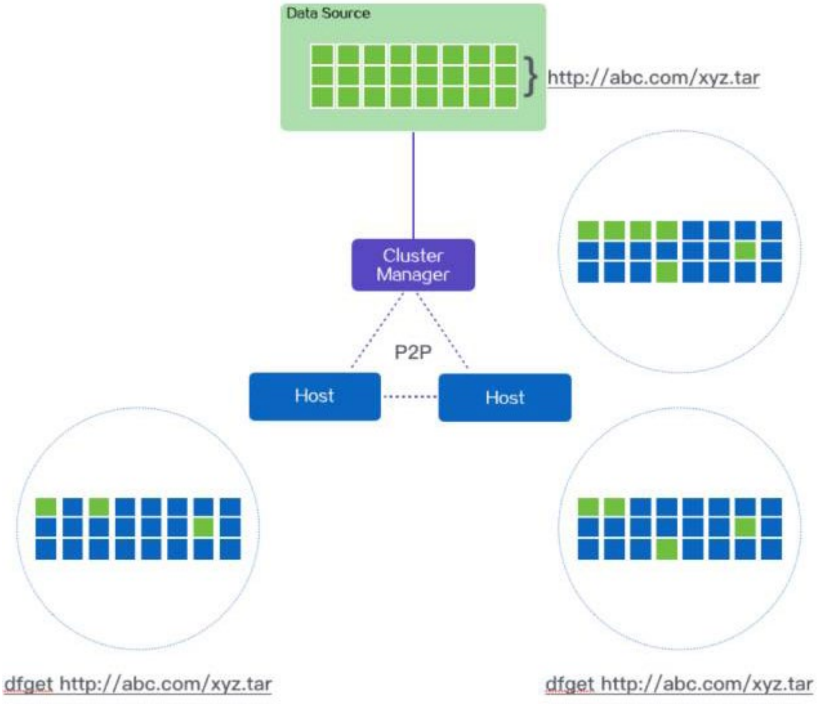
1. 以被动 CDN 方式从文件源下载文件并生成一组种子分块数据；
2. 构造 P2P 网络并调度每个 peer 之间互传指定的分块数据。

Host 上就存放着 dfget，dfget 的语法跟 wget 非常类似。主要功能包括文件下载和 P2P 共享等。

在阿里内部我们可以用 StarAgent 来下发 dfget 指令，让一组机器同时下载文

件，在某种场景下一组机器可能就是阿里所有的服务器，所以使用起来非常高效。除了客户端外，蜻蜓还有 Java SDK，可以让你将文件“PUSH”到一组服务器上。

下面这个图阐述了两个终端同时调用 dfget，下载同一个文件时系统的交互示意图：



蜻蜓 P2P 组网逻辑示意图

两个 Host 和 CM 会组成一个 P2P 网络，首先 CM 会查看本地是否有缓存，如果没有，就会回源下载，文件当然会被分片，CM 会多线程下载这些分片，同时会将下载的分片提供给 Host 们下载，Host 下载完一个分片后，同时会提供出来给 peer 下载，如此类推，直到所有的 Host 全部下载完。

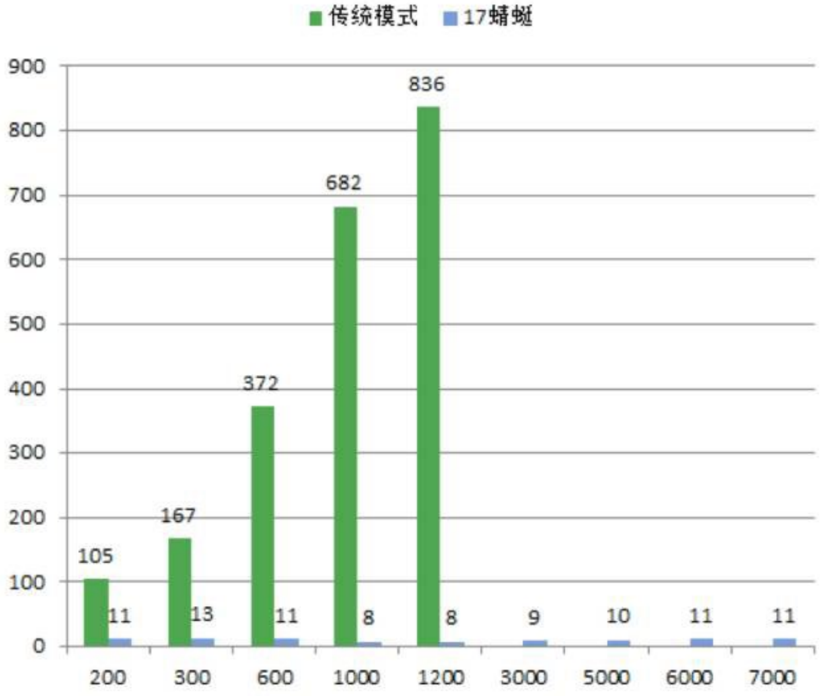
本地下载的时候会将下载分片的情况记录在 metadata 里，如果突然中断了下载，再次执行 dfget 命令，会断点续传。

下载结束后，还会比对 MD5，以确保下载的文件和源文件是完全一致的。蜻蜓

通过 HTTP cache 协议来控制 CM 端对文件的缓存时长，CM 端当然也有自己定期清理磁盘的能力，确保有足够的空间支撑长久的服务。

在阿里还有很多文件预热的场景，需要提前把文件推送到 CM 端，包括容器镜像、索引文件、业务优化的 cache 文件等等。

在第一版上线后，我们进行了一轮测试，结果如下图：



传统下载和蜻蜓 P2P 下载测试结果对比图

X 轴是客户端数量，Y 轴是下载时长，

文件源：测试目标文件 200MB（网卡：千兆 bit/s）

Host 端：百兆 bit/s 网卡

CM 端：2 台服务器（24 核 64G，网卡：千兆 bit/s）

从这个图可以看出两个问题：

1. 传统模式随着客户端的增加，下载时长跟着增加，而 dfget 可以支撑到 7000 客户端依然没变好。
2. 传统模式到了 1200 客户端以后就没有数据了，因为数据源被打爆了。

从发布系统走向基础设施

2015 年双 11 后，蜻蜓的下载次数就达到了 12 万 / 月，分发量 4TB。当时在阿里还有别的下载工具，如 wget，curl，scp，ftp 等等，也有自建的小规模文件分发系统。我们除了全面覆盖自身发布系统外，也做了小规模的推广。到 2016 年双 11 左右，我们的下载量就达到了 1.4 亿 / 月，分发量 708TB，业务增长了近千倍。

2016 年双 11 后我们提出了一个更高的目标，希望阿里大规模文件分发和大文件分发 90% 的业务由蜻蜓来承担。

我希望通过这个目标锤炼出最好的 P2P 文件分发系统。此外也可以统一集团内所有的文件分发系统。统一可以让更多的用户受益，但统一从来不是终极目标，统一的目的是：1. 减少重复建设；2. 全局优化。

只要优化蜻蜓一个系统，全集团都能受益。比如我们发现系统文件是每天全网分发的，而光这一个文件压缩的话就能给公司每天节省 9TB 网络流量。跨国带宽资源尤其宝贵。而如果大家各用各的分发系统，类似这样的全局优化就无从谈起。

所以统一势在必行！

在大量数据分析基础上，我们得出全集团文件分发的量大概是 3.5 亿次 / 周，而我们当时的占比只有 10% 不到。

经过半年努力，在 2017 年 4 月份，我们终于实现了这个目标，达到 90%+ 的业务占有率，业务量增长到 3 亿次 / 周（跟我们之前分析的数据基本吻合），分发量 977TB，这个数字比半年前一个月的量还大。

当然，不得不说这跟阿里容器化也是密不可分的，镜像分发流量大约占了一半。下面我们就来介绍下蜻蜓是如何支持镜像分发的。在说镜像分发之前先说下阿里的容器技术。

阿里的容器技术

容器技术的优点自然不需要多介绍了，全球来看，容器技术以 Docker 为主占了大部分市场，当然还有其他解决方案：比如 rkt，Mesos Uni Container，LXC 等，而阿里的容器技术命名为 Pouch。早在 2011 年，阿里就自主研发了基于 LXC 的容器技术 T4，只是当时我们没有创造镜像这个概念，T4 还是当做虚拟机来用，当然比虚拟机要轻量的多。

2016 年阿里在 T4 基础上做了重大升级，演变为今天的 Pouch，并且已经开源。目前 Pouch 容器技术已经覆盖阿里巴巴集团几乎所有的事业部，在线业务 100% 容器化，规模高达数十万。镜像技术的价值扩大了容器技术的应用边界，而在阿里如此庞大的应用场景下，如何实现高效“镜像分发”成为一个重大命题。

回到镜像层面。宏观上，阿里巴巴有规模庞大的容器应用场景；微观上，每个应用镜像在镜像化时，质量也存在参差不齐的情况。

理论上讲用镜像或者用传统“基线”模式，在应用大小上不应该有非常大的区别。但事实上这完全取决于 Dockerfile 写的好坏，也取决于镜像分层是否合理。阿里内部其实有最佳实践，但是每个团队理解接受程度不同，肯定会有用的好坏的之分。尤其在一开始，大家打出来的镜像有 3 ~ 4GB 这都是非常常见的。

所以作为 P2P 文件分发系统，蜻蜓就有了用武之地，无论是多大的镜像，无论是分发到多少机器，即使你的镜像打的非常糟糕，我们都提供非常高效的分发，都不会成瓶颈。这样就给我们快速推广容器技术，让大家接受容器运维模式，给予了充分消化的时间。

容器镜像

在讲镜像分发之前先简单介绍下容器镜像。我们看下 Ubuntu 系统的镜像：我们可以通过命令 docker history ubuntu:14.04 查看 ubuntu:14.04，结果如下：

```
root@DaoCloud:~# docker history ubuntu:14.04
IMAGE          CREATED       CREATED BY                                       SIZE
d2a0ecffe6fa   5 weeks ago   /bin/sh -c #(nop) CMD ["/bin/bash"]              0 B
29460ac93442   5 weeks ago   /bin/sh -c sed -i 's/^#\s*\(deb.*universe\)$/    1.895 kB
b670fb0c7ecd   5 weeks ago   /bin/sh -c echo '#!/bin/sh' > /usr/sbin/polic    194.5 kB
83e4dde6b9cf   5 weeks ago   /bin/sh -c #(nop) ADD file:c8f078961a543cdefa    188.2 MB
root@DaoCloud:~# 
```

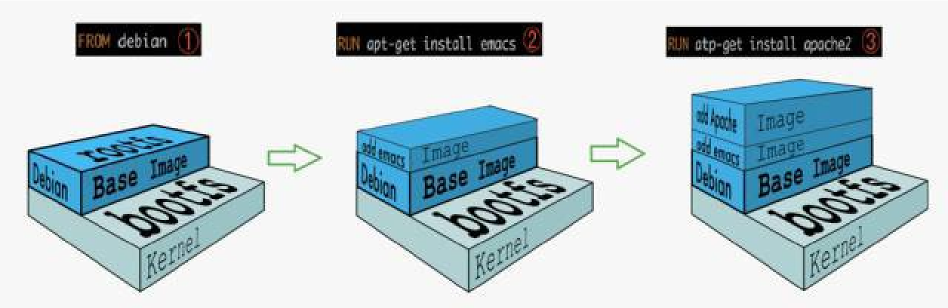
需要注意的是：镜像层 d2a0ecffe6fa 中没有任何内容，也就是所谓的空镜像。

镜像是分层的，每层都有自己的 ID 和尺寸，这里有 4 个 Layer，最终这个镜像是由这些 Layer 组成。

Docker 镜像是通过 Dockerfile 来构建，看一个简单的 Dockerfile：

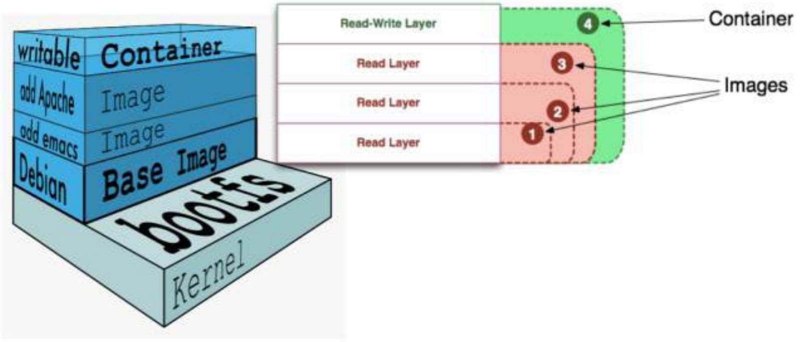
```
1. From Debian
2. RUN apt-get install emacs
3. RUN apt-get install apache2
4. CMD ["/bin/bash"]
```

镜像构建过程如下图所示：



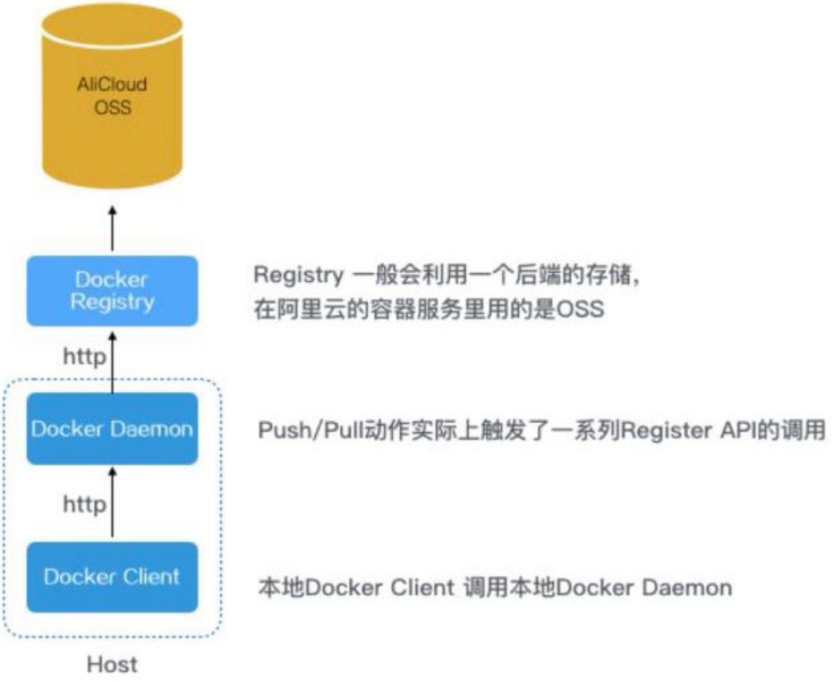
可以看到，新镜像是从 base 镜像一层一层叠加生成的。每安装一个软件，就在现有镜像的基础上增加一层。

当容器启动时，一个可写层会被加载到镜像的顶层，这个可读可写层也被称为“容器层”，容器层之下都是“镜像层”，都是只读的。



如果镜像层内容为空，相应的信息会在镜像 json 文件中描述，如果镜像层内容不为空，则会以文件的形式存储在 OSS 中。

镜像分发

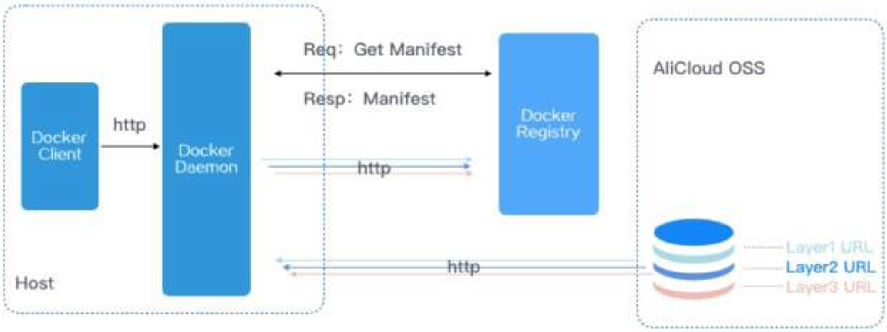


Docker 镜像下载流程图

以阿里云容器服务为例，传统的镜像传输如上图所示，当然这是最简化的一种架构模式，实际的部署情况会复杂的多，还会考虑鉴权、安全、高可用等等。

从上图可以看出，镜像传输跟文件分发有类似的问题，当有一万个 Host 同时向 Registry 请求时，Registry 就会成为瓶颈，还有海外的 Host 访问国内 Registry 时候也会存在带宽浪费、延时变长、成功率下降等问题。

下面介绍下 Docker Pull 的执行过程：



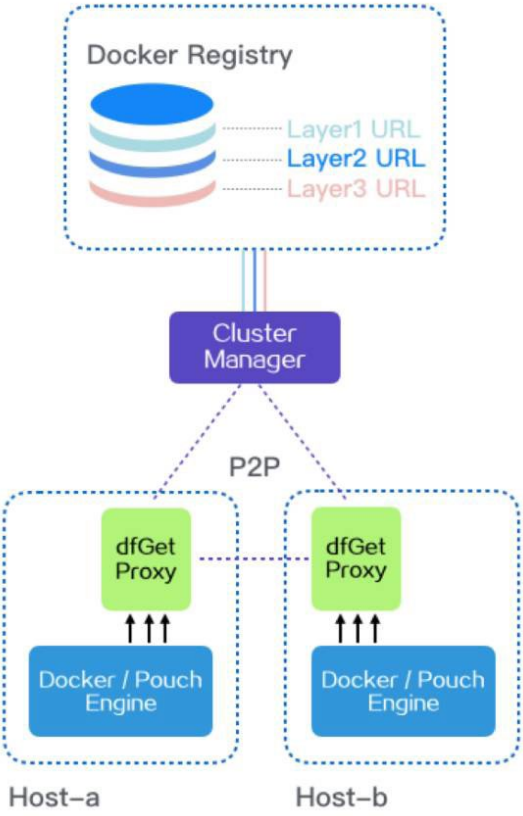
Docker 镜像分层下载图

Docker Daemon 调用 Registry API 得到镜像的 Manifest，从 Manifest 中能算出每层的 URL，Daemon 随后把所有镜像层从 Registry 并行下载到 Host 本地仓库。

所以最终，镜像传输的问题变成了各镜像层文件的并行下载的问题。而蜻蜓擅长的正是将每层镜像文件从 Registry 用 P2P 模式传输到本地仓库中。

那么具体又是如何做到的呢?

事实上我们会在 Host 上启动 dfGet proxy，Docker/Pouch Engine 的所有命令请求都会通过这个 proxy，我们看下图：



蜻蜓 P2P 容器镜像分发示意图

首先，docker pull 命令，会被 dfget proxy 截获。然后，由 dfget proxy 向 CM 发送调度请求，CM 在收到请求后会检查对应的下载文件是否已经被缓存到本地，如果没有被缓存，则会从 Registry 中下载对应的文件，并生成种子分块数据（种子分块数据一旦生成就可以立即被使用）；如果已经被缓存，则直接生成分块任务，请求者解析相应的分块任务，并从其他 peer 或者 supernode 中下载分块数据，当某个 Layer 的所有分块下载完成后，一个 Layer 也就下载完毕了，同样，当所有的 Layer 下载完成后，整个镜像也就下载完成了。

蜻蜓支持容器镜像分发，也有几个设计目标：

1. 大规模并发：必须能支持十万级规模同时 Pull 镜像。

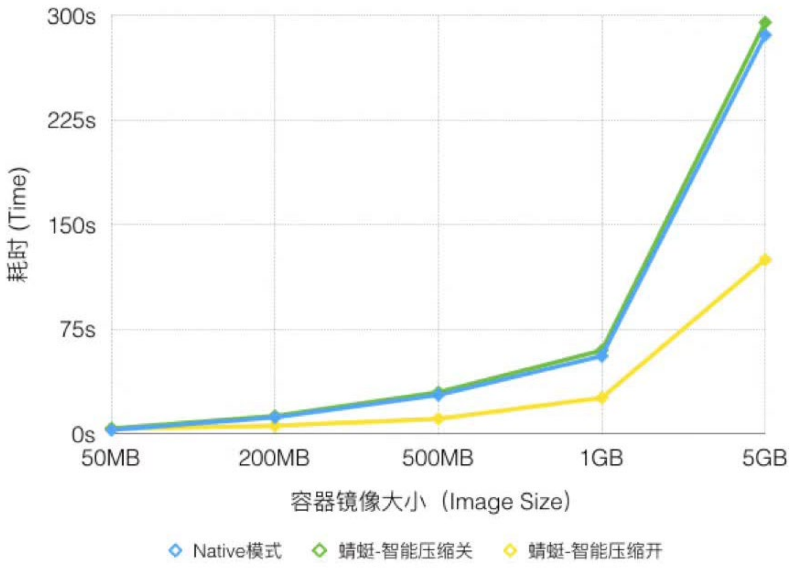
2. 不侵入容器技术内核（Docker Daemon，Registry）：也就是说不能改动容器服务任何代码。
3. 支持 Docker，Pouch，Rocket ，Hyper 等所有容器 / 虚拟机技术。
4. 支持镜像预热：构建时就推送到蜻蜓集群 CM。
5. 支持大镜像文件：至少 30GB。
6. 安全

Native Docker V.S 蜻蜓

我们一共做了两组实验：

实验一：1 个客户端

1. 测试镜像大小：50MB、200MB、500MB、1GB、5GB
2. 镜像仓库带宽：15Gbps
3. 客户端带宽：双百兆 bit/s 网络环境
4. 测试规模：单次下载

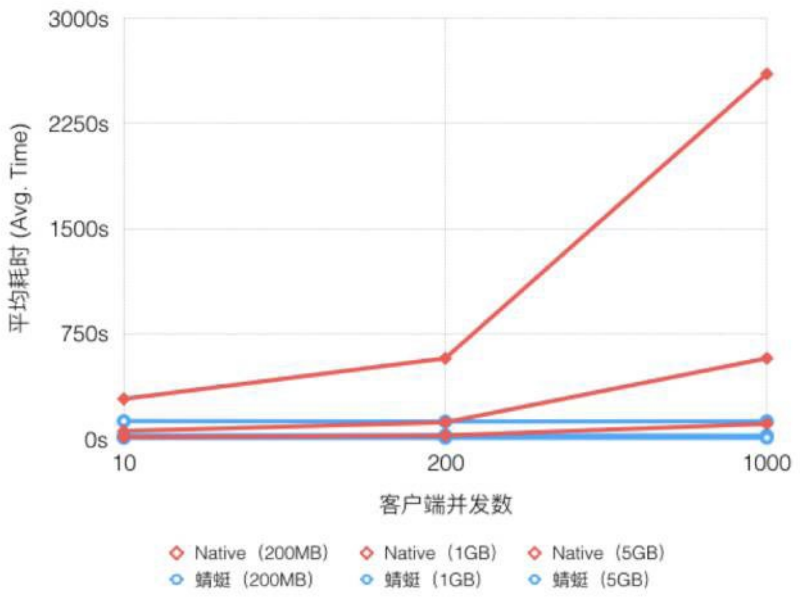


单客户端不同模式对比图

Native 和蜻蜓（关闭智能压缩特性）平均耗时基本接近，蜻蜓稍高一点，因为蜻蜓在下载过程中会校验每个分块数据的 MD5 值，同时在下载之后还会校验整个文件的 MD5，以保证下载的文件跟源文件是一致的；而开启了智能压缩的模式下，其耗时比 Native 模式还低！

实验二：多客户端并发

1. 测试镜像大小：50MB、200MB、500MB、1GB、5GB
2. 镜像仓库带宽：15Gbps
3. 客户端带宽：双百兆 bit/s 网络环境
4. 多并发：10 并发、200 并发、1000 并发



不同镜像大小和并发数的对比图

上图可以看出，随着下载规模的扩大，蜻蜓与 Native 模式耗时差异显著扩大，最高可提速可以达 20 倍。在测试环境中源的带宽也至关重要，如果源的带宽是 2Gbps，提速可达 57 倍。

下图是下载文件的总流量（并发数 * 文件大小）和回源流量（去 Registry 下载的

流量）的一个对比：

蜻蜓镜像分发出流量对比图

向 200 个节点分发 500M 的镜像，比 docker 原生模式使用更低的网络流量，实验数据表明采用蜻蜓后，Registry 的出流量降低了 99.5% 以上；而在 1000 并发规模下，Registry 的出流量更可以降低到 99.9% 左右。

阿里巴巴实践效果

蜻蜓在阿里投入使用大概已有两年，两年来业务发展迅速，从分发的次数来统计目前一个月接近 20 亿次，分发 3.4PB 数据。其中容器镜像的分发量接近一半。

蜻蜓在阿里文件 vs 镜像分发流量趋势图

在阿里最大的一次分发应该就是今年双 11 期间，要对上万台服务器同时下发 5GB 的数据文件。

走向智能化

阿里在 AIOps 起步虽然不是最早，但是我们近年来投入巨大，并在很多产品上有所应用。蜻蜓这个产品中有以下应用：

智能流控

流控在道路交通中很常见，比如中国道路限速规定，没有中心线的公路，限速为 40 公里 / 小时；同方向只有 1 条机动车道的公路，限速为 70 公里 / 小时；快速道路 80 公里；高速公路最高限速为 120 公里 / 小时等等。这种限速对每辆车都一样，显然不够灵活，所以在道路非常空闲的情况下，道路资源其实是非常浪费的，整体效率非常低下。

红绿灯其实也是流控的手段，现在的红绿灯都是固定时间，不会根据现实的流量来做智能的判断，所以去年 10 月召开的云栖大会上，王坚博士曾感慨，世界上最遥远的距离不是从南极到北极，而是从红绿灯到交通摄像头，它们在同一根杆上，但从来没有通过数据被连接过，摄像头看到的东西永远不会变成红绿灯的行动。这既浪费了城市的数据资源，也加大了城市运营发展的成本。

蜻蜓其中一个参数就是控制磁盘和网络带宽利用率的，用户可以通过参数设定使用多少网络 IO/ 磁盘 IO。如上所述，这种方法是非常僵化的。所以目前我们智能化方面的主要思想之一是希望类似的参数不要再人为来设定，而是根据业务的情况结合系统运行的情况，智能的决定这些参数的配置。最开始可能不是最优解，但是经过一段时间运行和训练后自动达到最优化的状态，保证业务稳定运行同时又尽可能的充分利用网络和磁盘带宽，避免资源浪费。

智能调度

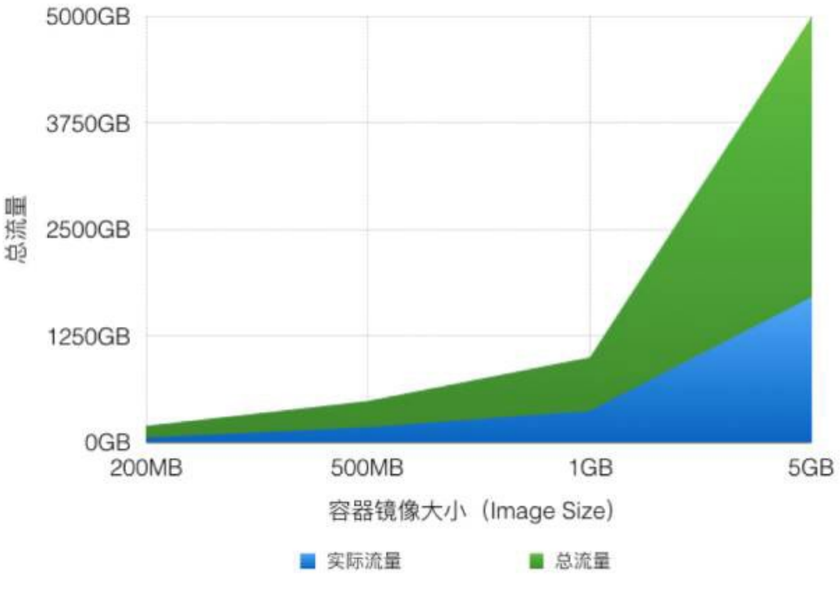
分块任务调度是决定整个文件分发效率高低与否的关键因素，如果只是通过简单的调度策略，比如随机调度或者其他固定优先级的调度，这种做法往往会引起下载速率的频繁抖动，很容易导致下载毛刺过多，同时整体下载效率也会很差。为了最优化

任务调度，我们经历了无数次的尝试和探索，最终通过多维度（比如机器硬件配置、地理位置、网络环境、历史下载结果和速率等等维度的数据）的数据分析（主要利用了梯度下降算法，后续还会尝试其他算法），智能动态决定当前请求者最优的后续分块任务列表。

智能压缩

智能压缩会对文件中最值得压缩的部分实施相应的压缩策略，从而可以节约大量的网络带宽资源。

对容器镜像目前的实际平均数据来看，压缩率 (Compression Ration) 是 40%，也就是说 100MB 镜像可以压缩到 40MB。针对 1000 并发规模，通过智能压缩可以减少 60% 的流量。



安全

在下载某些敏感的文件（比如秘钥文件或者账号数据文件等）时，传输的安全性必须要得到有效的保证，在这方面，蜻蜓主要做了两个工作：

1. 支持携带 HTTP 的 header 数据，以满足那些需要通过 header 来进行权限验证的文件源；
2. 利用对称加密算法，对文件内容进行传输加密。

开源

随着容器技术的流行，容器镜像这类大文件分发成为一个重要问题，为了更好的支持容器技术的发展，数据中心大规模文件的分发，阿里决定开源蜻蜓来更好的推进技术的发展。阿里将持续支持开源社区，并把自己经过实战检验的技术贡献给社区。敬请期待。

总结

蜻蜓通过使用 P2P 技术同时结合智能压缩、智能流控等多种创新技术，解决大规模文件下载以及跨网络隔离等场景下各种文件分发难题，大幅提高数据预热、大规模容器镜像分发等业务能力。

蜻蜓支持多种容器技术，对容器本身无需做任何改造，镜像分发比 natvie 方式提速可高达 57 倍，Registry 网络出流量降低 99.5% 以上。承载着 PB 级的流量的蜻蜓，在阿里已然成为重要的基础设施之一，为业务的极速扩张和双 11 大促保驾护航。

PS：云效 2.0 智能运维平台 – 致力于打造具备世界级影响力的智能运维平台，诚聘资深技术 / 产品专家，工作地点：杭州、北京、美国。

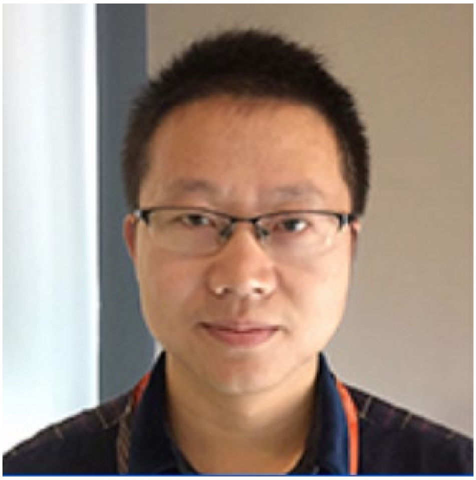
Reference

[1] Docker Overview：
https://docs.docker.com/engine/docker-overview/
[2] Where are docker images stored：
http://blog.thoward37.me/articles/where-are-docker-images-stored/
[3] Image Spec：
https://github.com/moby/moby/blob/master/image/spec/v1.md
[4] Pouch 开源地址：
https://github.com/alibaba/pouch
[5] 蜻蜓开源地址：

https://github.com/alibaba/dragonfly
[6] 阿里云容器服务：
https://www.aliyun.com/product/containerservice
[7] 飞天专有云敏捷版：
https://yq.aliyun.com/articles/224507
[8] 云效智能运维平台：
https://www.aliyun.com/product/yunxiao

双 11 万亿流量下的分布式缓存

宗岱



宗岱：阿里巴巴资深技术专家，2008 年加入淘宝，阿里分布式缓存、NoSQL 数据库 Tair 和 Tengine 负责人。

Tair 概览

Tair 发展历程

Tair 在阿里巴巴被广泛使用，无论是淘宝天猫浏览下单，还是打开优酷浏览播放时，背后都有 Tair 的身影默默支撑巨大的流量。Tair 的发展历程如下：

- 2010.04 Tair v1.0 正式推出 @ 淘宝核心系统；
- 2012.06 Tair v2.0 推出 LDB 持久化产品，满足持久化存储需求；
- 2012.10 推出 RDB 缓存产品，引入类 Redis 接口，满足复杂数据结构的存储需求；
- 2013.03 在 LDB 的基础上针对全量导入场景上线 Fastdump 产品，大幅度降低导入时间和访问延时；

- 2014.07 Tair v3.0 正式上线，性能 X 倍提升；
- 2016.11 泰斗智能运维平台上线，助力 2016 双 11 迈入千亿时代；
- 2017.11 性能飞跃，热点散列，资源调度，支持万亿流量。

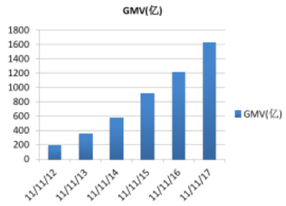
Tair 是一个高性能、分布式、可扩展、高可靠的 key/value 结构存储系统！Tair 特性主要体现在以下几个方面：

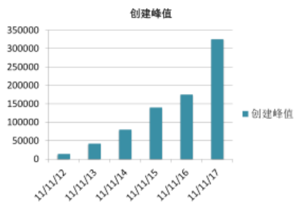
- 高性能：在高吞吐下保证低延迟，是阿里集团内调用量最大系统之一，双 11 达到每秒 5 亿次峰值的调用量，平均访问延迟在 1 毫秒以下；
- 高可用：自动 failover 单元化机房内以及机房间容灾，确保系统在任何情况下都能正常运行；
- 规模化：分布全球各个数据中心，阿里集团各个 BU 都在使用；
- 业务覆盖：电商、蚂蚁、合一、阿里妈妈、高德、阿里健康等。

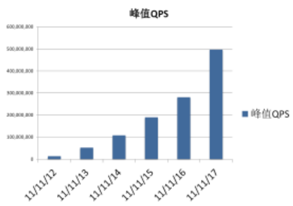
Tair 除了普通 Key/Value 系统提供的功能，比如 get、put、delete 以及批量接口外，还有一些附加的实用功能，使得其有更广的适用场景。Tair 应用场景包括以下四种：

1. MDB 典型应用场景：用于缓存，降低对后端数据库的访问压力，比如淘宝中的商品都是缓存在 Tair 中；用于临时数据存储，部分数据丢失不会对业务产生较大影响；读多写少，读 qps 达到万级别以上。
2. LDB 典型应用场景：通用 kv 存储、交易快照、安全风控等；存储黑白单数据，读 qps 很高；计数器功能，更新非常频繁，且数据不可丢失。
3. RDB 典型应用场景：复杂的数据结构的缓存与存储，如播放列表，直播间等。
4. FastDump 典型应用场景：周期性地将离线数据快速地导入到 Tair 集群中，快速使用到新的数据，对在线读取要求非常高；读取低延迟，不能有毛刺。

双 11 挑战怎么办?

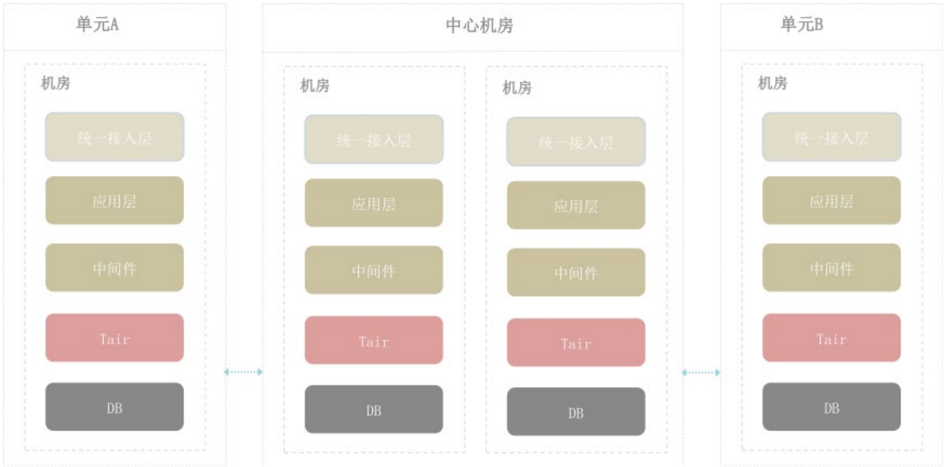






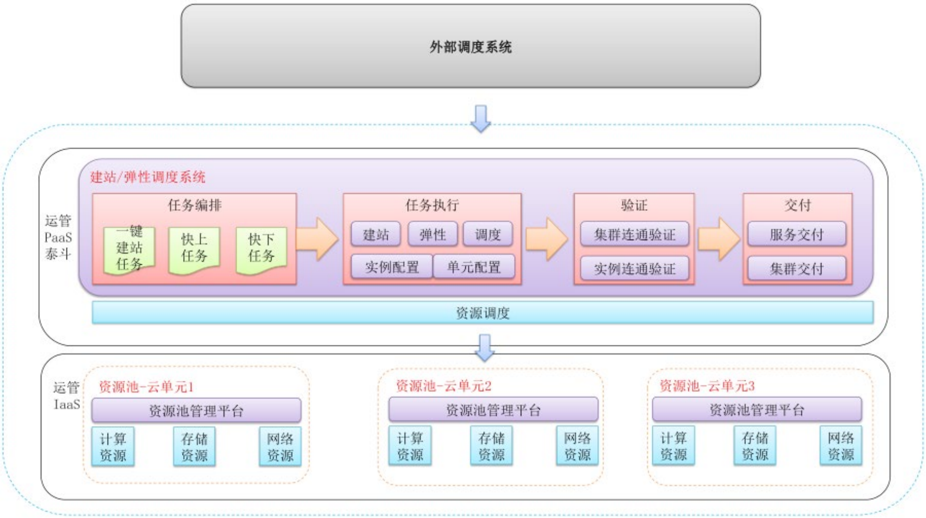
2012-2017 年数据如图，可以看到，2012 年 GMV 小于 200 亿，2017 年 GMV 达到 1682 亿，交易创建峰值从 1.4 万达到 32.5 万，峰值 QPS 从 1300 万达到近 5 亿。我们可以得出，Tair 访问峰值增速：Tair 峰值 > 交易峰值 > 总 GMV，如何确保成本不超过交易峰值增速？对于带数据的分布式系统来说，缓存问题都是比较难解决的，缓存流量特别大，2017 年双 11 后，我们彻底解决掉了缓存问题。我们现在的交易系统和导入系统都是多地域多单元多机房部署的，如何快速部署业务系统呢?

多地域多单元



多地域多单元首先是中心机房，我们做了双机房容灾，两侧还有若干个单元。机房内系统图如图，从流量接入进统一接入层 – 应用层 – 中间件 –Tair–DB，在数据层我们会做多地域的数据同步。

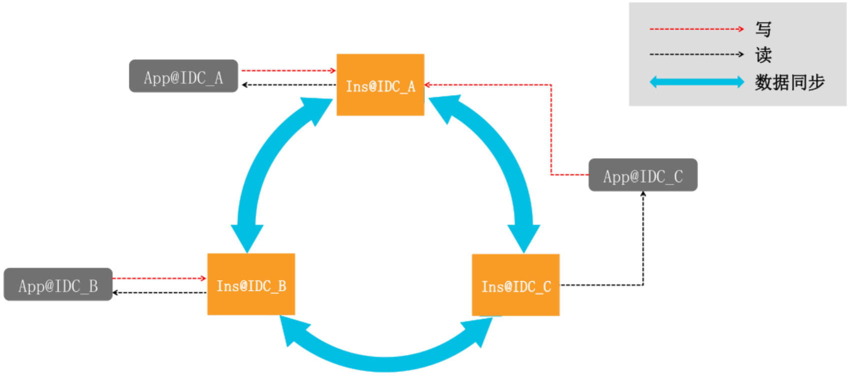
弹性建站



我们需要系统具备弹性建站的能力。Tair 是一个复杂的分布式存储系统，我们为之建立了一整套运营管控系统泰斗，在泰斗里建设弹性建站系统，它会经过一系列工作步骤将 Tair 系统建设起来，我们还会从系统层面、集群层面和实例连通层面进行验证，以确保系统功能、稳定性万无一失。

Tair 的每一个业务集群水位其实是不一样的，双 11 前的每一次全链路压测下，由于业务模型的变化，所用 Tair 资源会发生变化，造成水位出现变化。在此情况下，我们需要每一次压测完在多个集群间调度 Tair 资源，如果水位低，就会把某些机器服务器资源往水位高挪，达到所有集群水位值接近。

数据同步

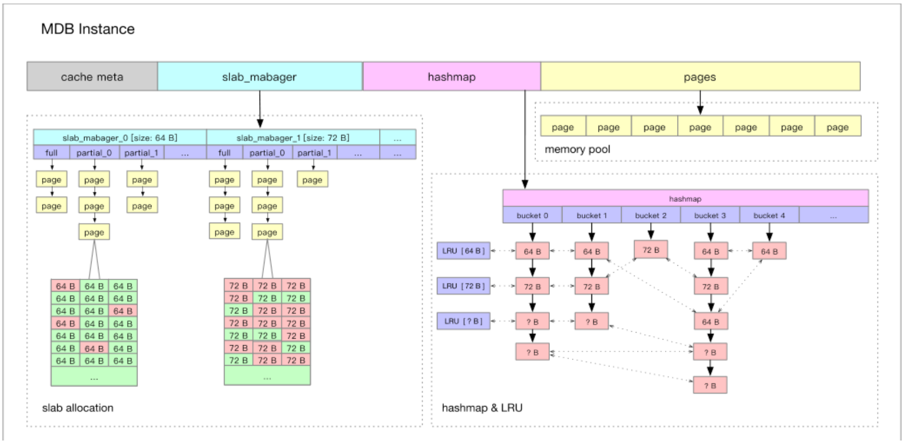


对于单元化业务，我们提供了本单元访问本地 Tair 的能力，对于有些非单元化业务，我们也提供了更灵活的访问模型。同步延迟是我们一直在做的事情，2017 年双 11 每秒同步数据已经达到了千万级别，那么，如何更好地解决非单元化业务在多单元写入数据冲突问题？这也是我们一直考虑的。

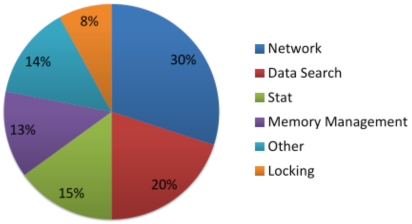
性能优化降成本

服务器成本并不是随着访问量线性增长，每年以百分之三四十成本在下降，我们主要通过服务器性能优化、客户端性能优化和不同的业务解决方案三方面达到此目的。

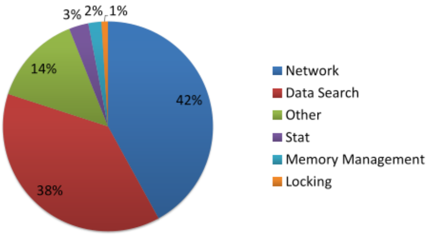
内存数据结构



图为 MDB 内存数据结构示意图，我们在进程启动之后会申请一大块内存，在内存中将格式组织起来。主要有 slab 分配器、hashmap 和内存池，内存写满后会经过 LRU 链进行数据淘汰。随着服务器 CPU 核数不断增加，如果不能很好处理锁竞争，很难提升整体性能。



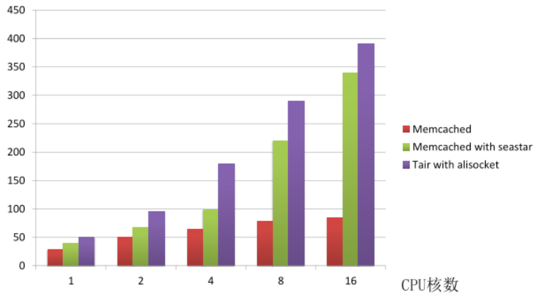
初始版本



优化版本

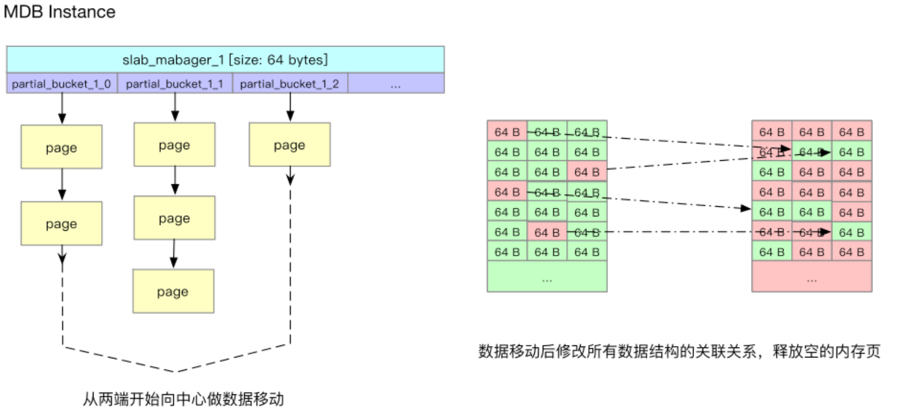
参考了各种文献和操作系统的设计，我们使用了细粒度锁、无锁数据结构、CPU 本地数据结构和读拷贝更新（读链表时不需要加锁）。左图为未经过优化时锁竞争各个功能模块消耗图，可以看到网络部分和数据查找部分消耗最多，优化后（右图）有 80% 的处理都是在网络和数据查找，这是符合我们期望的。

用户态协议栈



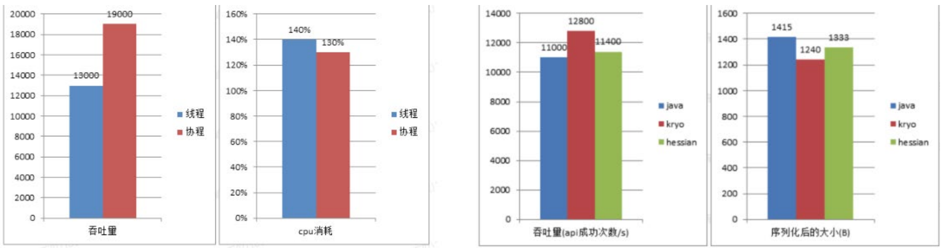
锁优化后，我们发现很多 CPU 消耗在内核态上，这时我们采用 DPDK+Alisocket 来替换掉原有内核态协议栈，Alisocket 采用 DPDK 在用户态进行网卡收包，并利用自身协议栈提供 socket API，对其进行集成。我们与业内类似系统进行了对比，如图。

内存合并



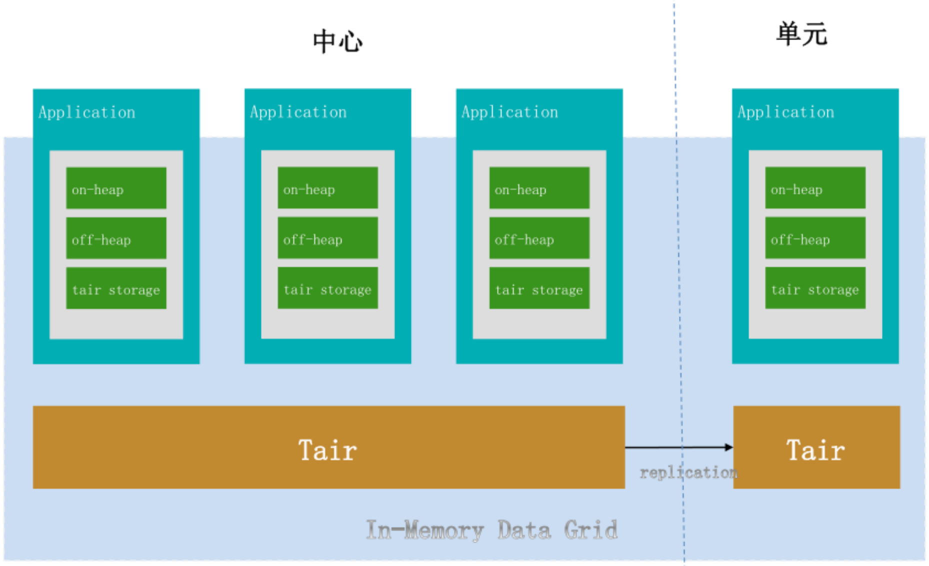
单机性能提升足够高后，带来问题是单位 qps 对应内存量变少了，怎么解决呢？我们发现公用集群整体看内存还是有的，某些 slab 没有办法写入，比如 64 字节 slab 没有被写满，但是 72 字节 slab 全部写满了，内存池都被申请完了，我们把 64 字节 slab 空闲页相互合并，这样可以释放大量空闲页。其中不可避免加入模式锁，在触发合并阈值情况下，切换成加锁状态，合并都是在访问量低峰期做的，对于业务峰值来说没有问题。

客户端优化



客户端我们做了两方面优化：网络框架替换，适配协程，从原有的 mina 替换成 netty，吞吐量提升 40%；序列化优化，集成 kryo 和 hessian，吞吐量提升 16%+。

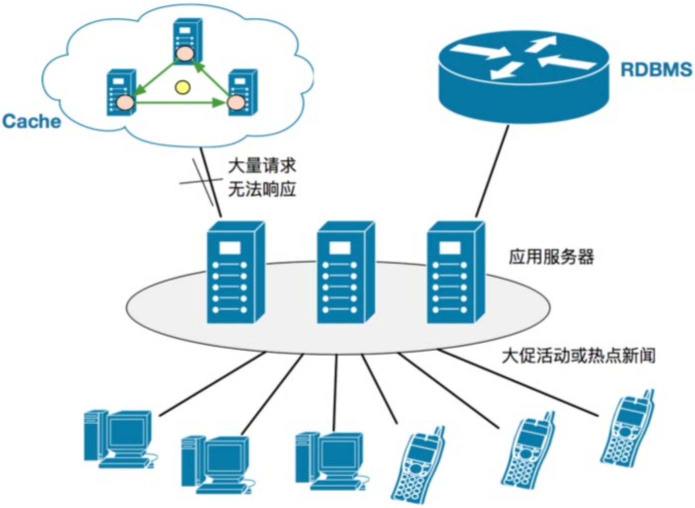
内存网格



如何与业务结合来降低整体 Tair 与业务成本？ Tair 提供了多级存储一体化解决业务问题，比如安全风控场景，读写量超大、有大量本地计算，我们可以在业务机器本地存下该业务机器所要访问的数据，大量读会命中在本地，而且写在一段时间内是可合并的，在一定周期后，合并写到远端 Tair 集群上作为最终存储。我们提供读写穿透，包括合并写和原有 Tair 本身具有多单元复制的能力，双 11 时业务对 Tair 读取降至 27.68%，对 Tair 写入降至 55.75%。

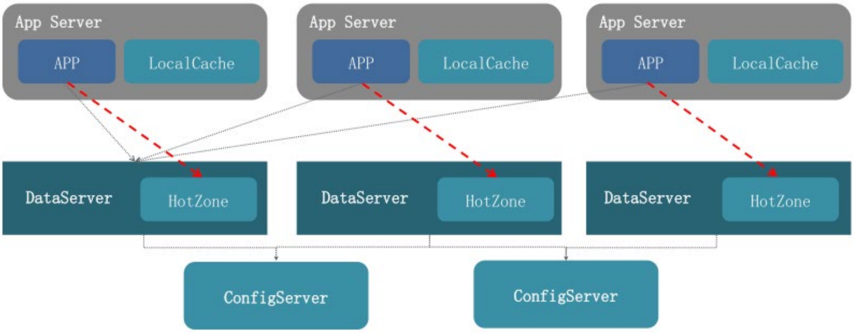
热点难题已解决

缓存击穿



缓存从开始的单点发展到分布式系统，通过数据分片方式组织，但对每一个数据分片来说，还是作为单点存在的。当有大促活动或热点新闻时，数据往往是在某一个分片上的，这就会造成单点访问，进而缓存中某个节点就会无法承受这么大压力，致使大量请求没有办法响应。即便是限流也是有损操作，可能也会造成全系统崩溃。我们发现，问题根源是访问热点，需要彻底解决该问题。

热点散列



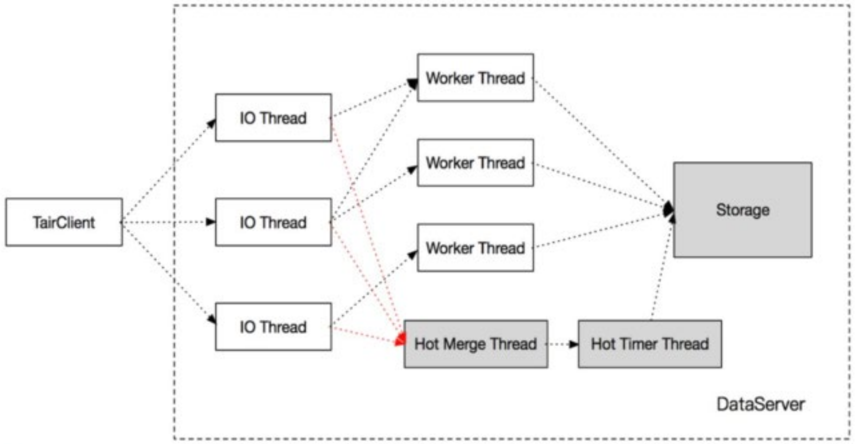
经过多种方案的探索，采用了热点散列方案。我们评估过客户端本地 cache 方案和二级缓存方案，它们可以在一定程度上解决热点问题，但各有弊端。而热点散列直接在数据节点上加 hotzone 区域，让 hotzone 承担热点数据存储。对于整个方案来说，最关键有以下几步：

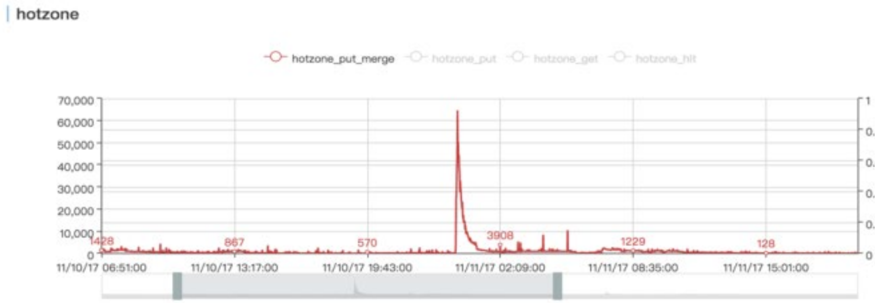
- 智能识别。热点数据总是在变化的，或是频率热点，或是流量热点。
- 实时反馈。采用多级 LRU 的数据结构，设定不同权值放到不同层级的 LRU 上，一旦 LRU 数据写满后，会从低级 LRU 链开始淘汰，确保权值高的得到保留。
- 动态散列。当访问到热点时，Appserver 和服务端就会联动起来，根据预先设定好的访问模型动态散列到其它数据节点 hotzone 上去访问，集群中所有节点都会承担这个功能。

通过这种方式，我们将原来单点访问承担的流量通过集群中部分机器来承担。

可以看到，双 11 零点的刹那，我们吸收了 800 多万次的热点访问。如果没有做热点散列，散列前的指数都会超过死亡水位线。

写热点





写热点与读热点有类似的地方，也需要把热点实时识别出来，然后通过 IO 线程把有热点 key 的请求放给合并线程去处理，会有定时处理线程提交给存储层。

经过双 11 考验对读写热点的处理，我们现在可以放心的说，我们将缓存包括 kv 存储部分的读写热点彻底解决了。

2017 双 11 交易系统 TMF2.0 技术揭秘，实现全链路管理

毗卢

阿里巴巴资深技术专家毗卢

毗卢，阿里巴巴资深技术专家，主导设计了 TMF2.0 框架，并基于该框架完成交易平台架构升级改造，目前负责商品中心，专注电商领域业务建模与工程交付相结合的研究与平台推广。

交易平台遇到的挑战

在刚刚过去的 2017 双 11，交易峰值达到了 32.5 万笔 / 秒，这给整个交易系统带来了非常大的挑战。一方面，系统需要支撑全集团几十个事业部的所有交易类需求：要考虑如何能更快响应需求、加快发布周期；如何能为新小业务提供快速支撑、降低准入门槛；是否足够开放使得业务方能做到自助式扩展；新需求是否已经在其他

事业部有可复用资产等问题。另一方面，整个电商体系涉及的应用高达 7000+：要考虑需求的评估是否具有全链路视角；业务需求的技术评估是否分析全面、技术方案的影响范围是否评估到位；业务的全链路稳定性保障、调用链路监控、强弱依赖等问题。此外面对每天几百个业务需求，500+ 个独立的发布变更：要考虑各业务方的需求发布是否会相互产生影响；需求代码是否对平台有侵入、导致平台腐化；高频率的需求发布下如何管控质量；能否按业务维度进行业务监控、故障分析等等。

TMF2.0 解决的关键问题

面对这些挑战，TMF2.0 框架需要六大关键问题。

- 业务可视化：平台能力、业务规则决定是否对外透出；
- 需求结构化支持：基于透出的业务能力、已有的业务规则完成需求结构化分解降低沟通成本；
- 业务配置化：这是可视化的前提，要在需求明确的情况下在线配置业务、快速发布上线；
- 业务测试一体化：根据修改的代码进行自动化用例筛选、自动化测试；
- 业务监控：以精细化的业务维度进行监控，而不仅仅局限于交易大盘；
- 故障排查：当业务故障时快速拿到故障快照、还原故障现场以及迅速定位问题原因。

针对以上六大关键问题，TMF2.0 的关键设计点有以下三个层面。

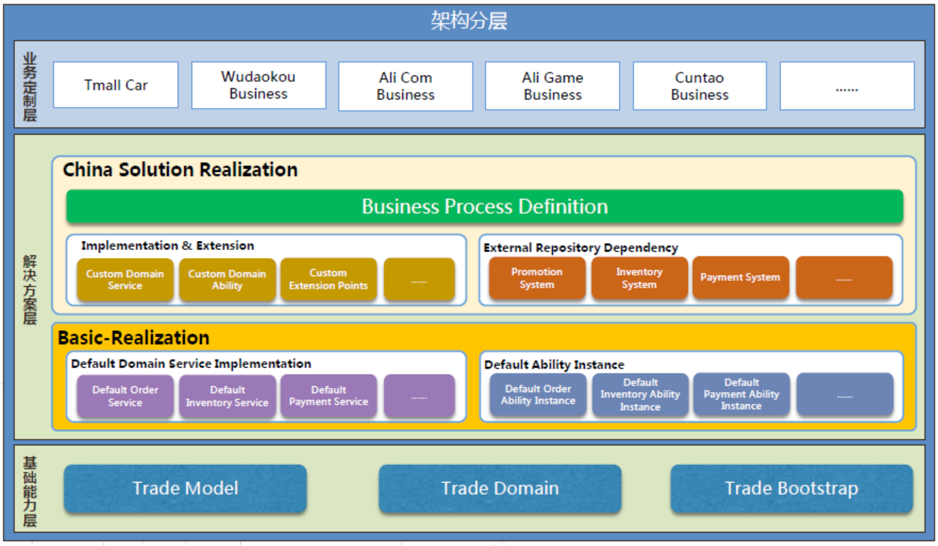
首先，需要实现业务 / 平台分离插件化架构。平台提供插件包注册机制，实现业务方插件包在运行期的注册。业务代码只允许存在于插件包中，与平台代码严格分离。业务包的代码配置库也与平台的代码库分离，通过二方包的方式，提供给容器加载。

其次，要统一业务身份。平台需要能有按“业务身份”进行业务与业务之间逻辑隔离的能力，而不是传统 SPI 架构不区分业务身份，简单过滤的方式。如何设计这个业务身份，也成为业务与业务之间隔离架构的关键。

另外，要注重管理域与运行域分离。业务逻辑不能依靠运行期动态计算，要能在静态期进行定义并可视化呈现。业务定义中出现的规则叠加冲突，也在静态器进行冲突决策。在运行期，严格按照静态器定义的业务规则、冲突决策策略执行。

下文将针对这三块的内容分别展开来详细介绍。

业务定制包与平台分离的架构



如上所示的业务定制包与平台分离架构可以分为四个层次。最底层是交易规范层，包括一些交易模型、交易领域的划分、业务领域的划分、以及交易启动环境下的配置项。基于这个理论模型，就可以进行一些定义及规范工作，比如接口定义、流程规范、模型规范等，而且其中的很多内容都可以在不同的领域进行复用。

上面一层是解决方案层。大家都知道阿里巴巴目前正在走国际化的战略，所以面对不同的市场会构建不同的解决方案，不同的解决方案中也就有自己不同的业务玩法、业务逻辑。所以要将不同的市场解决方案和他们自身的流程、规则结合起来。但是这一过程中会发现，不同的市场解决方案会有很多可以复用的地方，比如营销模式。所以形成的可复用基础实现就可以在不同的解决方案中得到复用，所那么在面对不同的市场时就不用考虑可复用基础实现的内容，只需要关注市场相关的业务就可以了。

往上一层是业务定制层。即使是在一个市场内，也会有各种细分的定制玩法，这些不同的细分点就会有各自不同的业务逻辑，这就是制定业务定制层的原因。团队会根据底层的需求点来进行一些业务定制包的组装，就可以实现不同的业务逻辑和玩法了。

在这样一个复杂的分离架构中，最重要的是要将不同层次间的职责划分清晰，整个代码都严格地、有意识地进行分离。所以在最后的部署过程中，首先要完成底层业务的复用，然后形成不同市场的解决方案，再在解决方案下对不同的业务实现差异化的点。

业务身份定义标准化

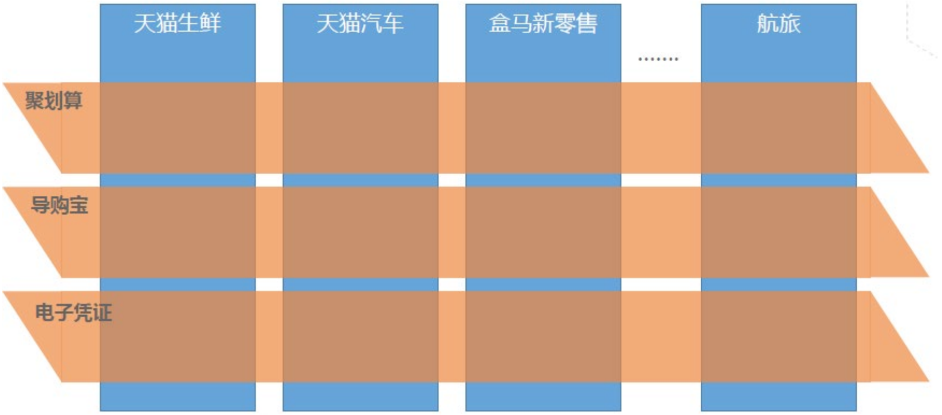
上面所讲的是业务和平台的分离，在业务和平台分离之后就要进行业务和业务之间的隔离，即统一的业务身份，类似于身份证号码，在整个交易链路上必须是唯一的。业务身份需要通过人、货、场三个维度进行抽象，比如市场类型、垂直市场、渠道来源等等，确定了这个唯一的业务身份后就可以将业务流程和业务规则进行关联。

基于业务识别，团队也提供了一个基于 UIL 的业务身份识别方案，总体设计基于标准模型来抽象，自定义语法，统一管理模型。事实上，通过样品模型、买家模型、卖家模型、类目模型这四个维度，99% 的商品都可以有效地进行标识。业务身份确定后，就可以按照业务身份维度，对业务配置、部署进行统一管理，在这其中要注意配置隔离性、热部署、配置回滚、配置确定性等核心要素。

业务管理域与运行域分离的框架

业务身份确定后就要进行业务定义，这其中就涉及管理域和运行域分离的问题。管理域就是指对业务生命周期、业务身份、业务对象进行定义，包括业务流程、业务管理等。这些操作完成之后就会将配置文件下发到，运行域上的各种平台就会自动解析配置域所下发的配置文件，然后将配置文件解析成业务命令来执行。

在上面所讲的业务域中，一个核心的问题就是如何定义业务：核心三要素是业务身份、业务叠加关系、冲突决策，即基于业务协议标准定义业务，执行单元按协议执行业务逻辑。



在业务叠加关系中，业务的复杂度就在于业务规则在不同维度下产生的冲突。业务的复杂度可以分为两个维度，一个是横向维度，一个是垂直维度。

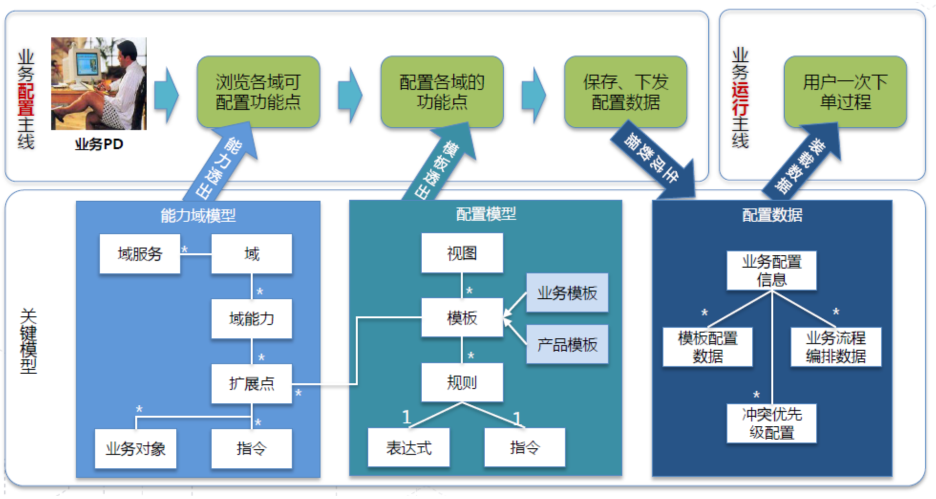
垂直维度，也可称之为“行业”。往往一个特定的“业务对象”（如商品），在静态期就能确认其具体归属于哪个行业。行业与行业之间的业务规则是不会有叠加的。比如，付款超时时间，各可以都设置为 1 天超时。但“天猫汽车”把超时时间改了，一定不会联动改其他业务的超时设置。横向维度，也称为产品维度，特点有：产品是可以被多个垂直业务所使用的、一个垂直业务是可以使用多个产品的、产品是否生效是需要结合业务会话的。比如，“电子凭证”是否生效，要看用户是否选择了“电子凭证”的交付方式。

通过业务复杂度的分析，可以得出一个结论是：一次业务会话完整的规则 =1 个垂直业务规则集合 + N 个水平业务规则集。所以在做业务定义和管理的时候，具体就

是在管某一个垂直业务是和哪些横向业务在叠加。在叠加之后产生的业务冲突又是怎么解决的？要基于这一点进行业务管理。这是比较关键的一点。

TMF 2.0 的关键模型介绍

基于以上的业务域介绍，下面详细阐述一下 TMF 2.0 的关键模型，主要包括业务配置主线和业务运行主线。



在业务配置主线中，由项目的业务 PD 来看一下当前业务涉及到哪些业务域，以及这些业务域下面有哪些功能和产品可以去使用，哪些业务点是可以去扩展的。这其中就需要能力域模型的支撑，通过这个模型所透出的结构化数据，来研究平台中每个域具备的能力、每个能力具有的可变点，从而有针对性地进行设置。在配置模型里，通过关键的视图模板，进行模板透出，然后保存、下发配置数据到业务运行主线。业务配置主线和业务运行主线是相交互的。

基于 TMF 2.0 关键模型，整个交易平台实现了业务定义可视、可管、可配。业务定义可视化包括系统能力可视化、业务流程可视化、业务规则可视化、产品叠加可视化等；业务可配置，所见即所得的业务规则可配置能力，凡是基于 TMF2 标准构建的系统均立刻可获取业务可配置能力，不需做额外的开发；配置版本化，针对业务配置有完善的版本化管理机制，配置推送可实现按版本快速生效或者回退；业务多租

户管理，不同的业务系统之间可以通过租户完全隔离的。不同的租户有自己的数据空间，以及配置推送策略。

在实际应用中，基于 TMF2.0 交易平台改造效果具体如下：

- 业务需求平均开发周期缩短至 12 天。比如汽车 4S 服务中，在老系统上做了一个月（未完成），新系统 7 天完成；五道口业务中，在老系统中评估工作量两个月，新系统 12 个工作日完成；饿了么业务中，老系统评估要两周，基于新系统 2 天完成。
- 平台与业务解耦。目前已完成的业务，其业务定制均只存在于业务包；在平台未改动情况下，业务方的发布更加灵活（有多次单业务发布，不需要其他业务方进行回归的案例）。
- 业务资产库。积累形成了 50+ 业务资产库，新业务可快速进行快速复制、调整并发布。

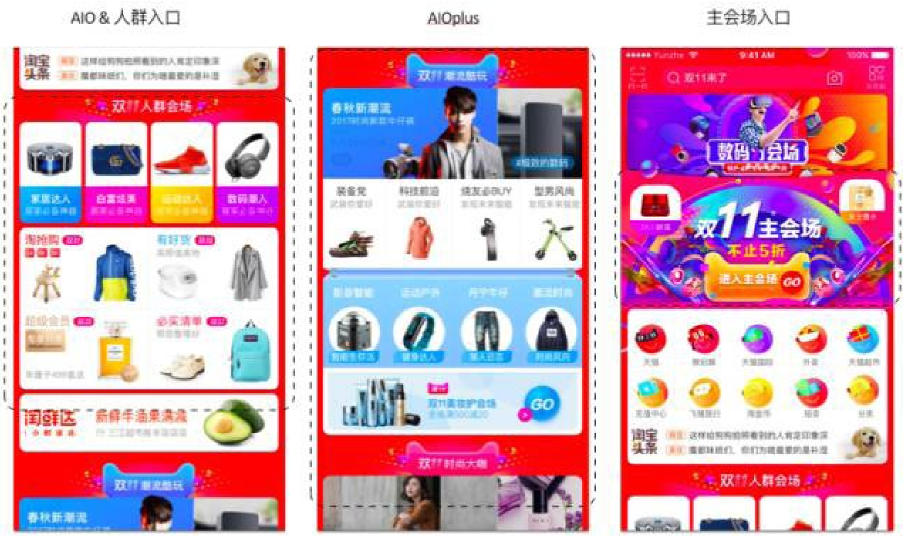
新体验

一天造出 10 亿个淘宝首页，阿里工程师如何实现？

灵培

阿里妹导读：双十一手淘首页个性化场景是推荐生态链路中最大的场景之一，在手淘 APP 承载了整体页面的流量第一入口，对用户流量的整体承接、分发、调控，以及用户兴趣的深度探索与发现上起着至关重要的作用。

双 11 手淘首页的几个重要推荐场景截图如下：



如上图所示，左一场景为 AIO 综合会场，包括 AIO 日常场景（淘抢购、有好货、清单等）、双 11 人群会场及行业会场；中间为 AIOplus 场景卡片综合会场，包括 5 张会场卡片，每张卡片融入了行业主分会场、标签会场，该业务涉及到 20 多个日常业务以及标签、行业会场的分发；右三为主会场入口所见所得，用两个素材轮播的方式给双 11 主会场进行引流。双十一当天整体点击 UV 引流效果方面，首页给各会场

还是取得了很好的分发效果，数据达到数千万 UV 以上。

与此同时，今年双十一在推荐的去重打散及探索发现上做了很多深度的优化，过去更多的是在相似性推荐的单一数据目标上进行优化，今年在 match 及 rank 技术上采用了更多多阶游走及探索发现的 embedding 技术，力争在 ctr 效果有一定保证的情况下，加大对用户体验比如多样性、搭配潜在兴趣、深度用户偏好等方面的推荐。

举个简单例子，之前的推荐系统在捕捉到用户对茶杯这一商品感兴趣后，很可能会推出更多的相似茶杯，新的推荐系统在多阶召回技术的基础上通过对用户兴趣进行深度学习的挖掘，会按一定的概率推荐茶叶、茶具等 " 弱相似 " 但满足用户搭配潜在兴趣的商品。

究竟阿里如何使用 AI 构建淘宝首页？今天一起来揭秘。

一、业务技术简介

首页个性化在算法技术上主要涉及 Graph Embedding 召回模型、Deep-Cross&ResNet 实时网络排序模型，并在搜索工程 Porsche&Blink、Rank Service、Basic Engine 等系统的基础上结合业务应用的需求沉淀了 Graph Embedding 召回框架及 XTensorflow 排序模型平台供推荐其他场景使用，提升效果均达到两位数以上。

二、首页个性化推荐框架（包括 MATCH 召回和 RANK 排序两部分）

1. 万物皆向量——Graph Embedding 深度召回框架

在推荐系统的发展历程中，面临了两个核心问题，用户的长尾覆盖度以及新商品的冷启动，这两个维度的数据扩展性瓶颈一直以来对广大推荐算法工程师都是不小的挑战。而我们基于 Graph Embedding 的理论知识提出的相关创新框架在召回阶段利用用户的序列化点击行为构建全网行为 graph，并结合深度速随机游走技术对用户行为进行 " 虚拟采样 " 拟合出多阶（一般 5 以上）的潜在兴趣信息，扩大用户的长尾兴趣宝贝召回，并同时利用 side information-based 的深度网络进行知识泛化学习，

在一定程度上解决了用户覆盖、新商品面临的冷启动问题，同时虚拟样本的采样技术结合深度模型的泛化学习等在用户对商品的探索发现上加大的扩大了召回量，提升了多样性及发现度。

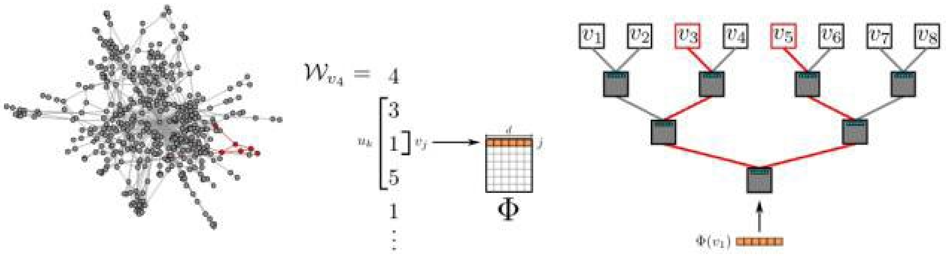
Graph Embedding 是一种将复杂网络投影到低维空间的机器学习算法，典型的做法是将网络中的节点做向量化表达，使节点间的向量相似度接近原始节点间在网络结构、近邻关系、meta 信息等多维度上的相似性。淘宝个性化推荐场景所面对的数以十亿计的用户、商品、交互数据和各类属性构成了一个规模庞大的异构网络，如果能将网络中的各类信息统一建模在同一个维度空间，用向量的方式进行表达，它的简洁和灵活性会有巨大的应用空间，诸如扩展 I2I 计算、解决商品冷启动、作为中间结果输出到上层高级模型。据我们所知，业界尚未有，对如此大规模复杂网络进行 embedding 建模的成熟应用。

本篇主要介绍我们近期在这个方向上所做的一些探索：针对推荐场景，在 Graph Embedding 基础上，提出了新的 S3 Graph Embedding Model 对上亿级别的商品进行 embedding 建模，并将 embedding 结果应用在商品 Item to Item 计算中，作为一种全新的 match 召回方式在手淘首图个性化场景进行应用。从线上 BTS 结果来看我们改进的 Graph Embedding I2I 得到不错的效果提升，在覆盖长尾用户以及新宝贝的冷启动上有效扩展了 match 召回候选。

1.1 Graph Embedding–DeepWalk 算法

Graph Embedding 是近期热门的一个课题，14 年 KDD 的《DeepWalk: Online Learning of Social Representations》开启了这个方向的热潮，文中借鉴了深度学习在语言模型中的应用，以全新的方式学习网络节点的潜在向量表示，在社会化网络多标签网络分类任务中取得了很好的效果。

DeepWalk 是一个 two-stage 算法：



(a) Random walk generation. (b) Representation mapping. (c) Hierarchical Softmax.

①构建同构网络，从网络中的每个节点开始分别进行 Random Walk 采样，得到局部相关联的训练数据；②对采样数据进行 SkipGram 训练，将离散的网络节点表示成向量化，最大化节点共现，使用 Hierarchical Softmax 来做超大规模分类的分类器；

- DeepWalk 框架：

Algorithm 1 DeepWalk(G, w, d, γ, t)

Input: graph $G(V, E)$
window size w
embedding size d
walks per vertex γ
walk length t

Output: matrix of vertex representations $\Phi \in \mathbb{R}^{|V| \times d}$

Initialization: Sample Φ from $\mathcal{U}^{|V| \times d}$
Build a binary Tree T from V
for $i = 0$ to γ **do**
  $\mathcal{O} = \text{Shuffle}(V)$
  for each $v_i \in \mathcal{O}$ **do**
    $\mathcal{W}_{v_i} = RandomWalk(G, v_i, t)$
    SkipGram(Φ, $\mathcal{W}_{v_i}$, w)
  end for
end for

- SkipGram 训练：

Algorithm 2 SkipGram(Φ, $\mathcal{W}_{v_i}$, w)

for each $v_j \in \mathcal{W}_{v_i}$ **do**
  for each $u_k \in \mathcal{W}_{v_i}[j-w : j+w]$ **do**
    $J(\Phi) = -\log \Pr(u_k \mid \Phi(v_j))$
    $\Phi = \Phi - \alpha * \frac{\partial J}{\partial \Phi}$
  end for
end for

首先从网络中采样训练数据，每一个训练数据是由局部相邻的节点组成的序列，DeepWalk 将这组序列看成语言模型中的一个短句或短语，将短句中的每个词转换成隐式表达，同时最大化给定短句某个中心词时，出现上下文单词的概率，具体可以表示为下面这个公式：

$$\underset{\Phi}{\text{minimize}} \quad -\log \Pr\left(\{v_{i-w}, \cdots, v_{i+w}\} \setminus v_i \mid \Phi(v_i)\right)$$

其中 v_i 是中心词 (对应于网络中的 target node)，v_(i−w),⋯,v_(i+w) 是上下文单词 (对应于网络中的 N 阶近邻的 node)。在独立分布的假设下，可以简化为：

$$\Pr\left(\{v_{i-w}, \cdots, v_{i+w}\} \setminus v_i \mid \Phi(v_i)\right) = \prod_{\substack{j=i-w \\ j \neq i}}^{i+w} \Pr(v_j | \Phi(v_i))$$

1.2 S³ Graph Embedding Model

针对推荐场景，我们将原本的 Graph Embedding 进行了多个方向上的创新改造，历经多个版本逐步演化出 S³ Graph Embedding Model，其中 S³ 主要体现在三个方面：

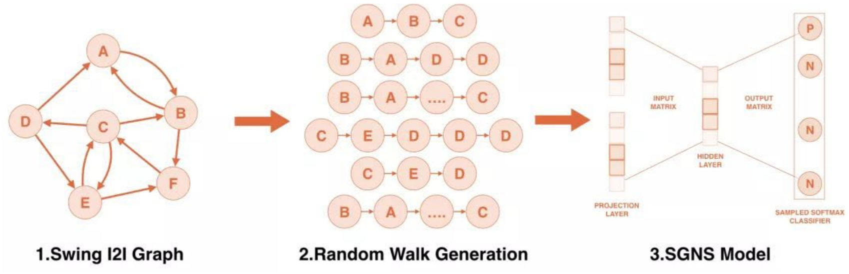
- Sequence Behavior
- Sub-graph

- Side-information

下面我们对演化的过程进行介绍：

（1）Naive 版本

- 将 Graph Embedding 应用到商品推荐领域中，第一步要解决的就是构建网络，由于商品推荐的整个网络是大规模的异构网络，网络节点类型包括用户、商品、商品类别、用户画像等各种基础属性，不同类型的节点无法在统一的空间进行向量表示。为了简化问题，首先使用阿里原创算法 swing 来计算商品间的相似度（swing 利用 user-item 二部图计算 i2i，比改进过的 CF 算法仍有明显提升，目前已在各场景中已经广泛应用，构建关于商品的有向带权图同构网络；
- 第二步是对商品网络进行 Random Walk 采样，此处我们借鉴了《Node-2vec: Scalable Feature Learning for Networks》中灵活的 BFS and DFS search strategies（在采样时以一定的几率回溯），通过改变控制参数灵活的控制 Random Walk 游走过程，在 Item Graph 的局部稳定性和全局扩展性间做出权衡，调节 match 结果中多样性和准确性之间的 balance。
- 在采样得到 item 序列之后，第三步是进行 Embedding 表示学习，这里对原文的 SkipGram 模型进行了两处优化：（1）针对超大规模的字典，采用负样本采样（Negative Sampling）替代 Hierarchical Softmax 分类器，大幅提高训练速度；（2）在语言学中，词按照出现频率从大到小排序之后，服从 Zipfian 分布（即 log-uniform 分布），所以 SkipGram 训练多直接采用 Zipfian 来计算负样本采样的概率，而对我们的数据进行分析后，发现 item 出现的频次与 log-uniform 并不完全吻合，不同时间段的数据间也呈现一定的波动趋势，针对这个现象，我们用边训练边统计每个 item 的出现频次，维护一个能够按照 item 出现频次生成负样本的动态采样器。模型结构如图所示：



可以看到，相比于 swing 的一阶扩展，Graph Embedding 的相似度计算将高阶信息也纳入其中。

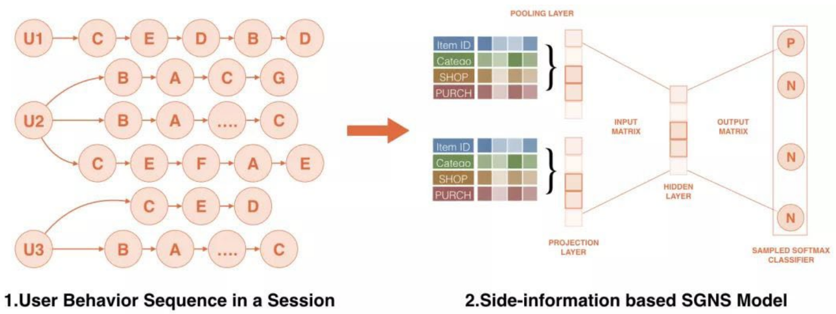
（2）Sequence+Side-information 版本

通过人工看 case 和统计得到的初步结论是基于已有 Swing 图计算 Graph Embedding 再召回的结果与原本的召回方式有较高的重复度，这在一定程度上给了我们信心，表明 embedding 的方式是靠谱的，为了在丰富多样性和提高精准度上有所突破，我们接下来做了三方面的尝试：

（1）通过用户 session 内的行为序列直接进行 embedding 建模；

（2）在 session 内行为构建全网图后，引入类似 tf-idf 的转移概率连接边，克服哈利波特热点问题，且在此基础上进行概率采样，构建用户行为的 " 虚拟样本 "，以扩大后面输入到深度模型里面的宝贝覆盖量及准确度，使多阶扩展信息更加完善；

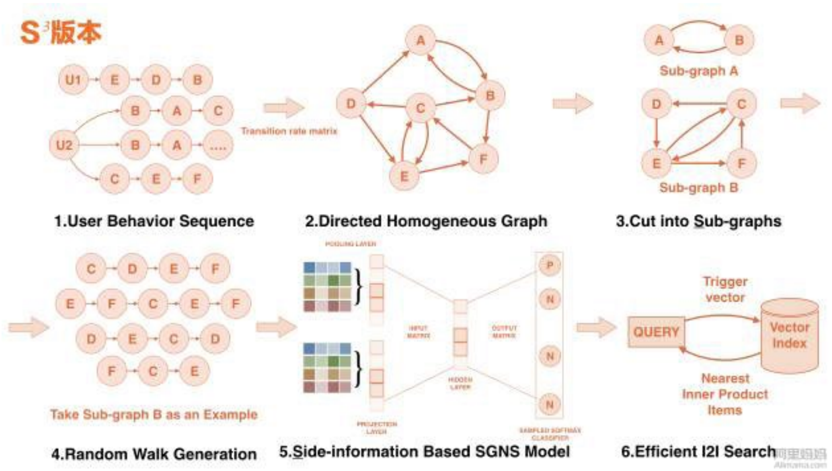
（3）引入 side-information（如一级类目、叶子类目、店铺、品牌、材质、购买力等级等）多维信息，通过 shared-embedding 加 pooling 的结构嵌入到 item 语义中。模型结构如图所示：



- 对于行为序列，短时间内的高密度点击更能够反映用户的真实意图，短期的兴趣偏好也更为集中，对比而言，基于用户点击共现的传统 CF 算法在数据层面会有更多的噪声。在实践中，准备行为序列数据有两个关键点值得注意：（1）如何选取行为序列的时间窗口，需要结合业务特点来进行仔细的调节；（2）如何过滤因为用户误操作而带来的噪声点，我们的实践经验是根据停留时长和类目不相似度来过滤典型的异常点击
- 提出并定义了新的转移概率连接图，是为了克服用户真实的 session 行为中存在的大量节点热点问题，以每个节点为中心，利用其扩散子节点的连接频次及行为共现频次计算转移概率，构建全网的转移概率连接图，并在此基础上进行 deep walk 的随机采样，构造出千亿级别的多阶虚拟样本，用于后续的深度网络学习
- 引入宝贝的 side-information 信息则受到语言模型中 sub-word 概念的启发，可类比于在 word2vector 中加入单词的词根、词缀等 sub-word 信息，认为某些具有同样词根词缀的单词在某些维度上具有相似的语义，具有同店、同品牌的宝贝也有一定的相关性。引入 side- information 后 i2i 结果的头部精度得到提高，同店、同品牌的宝贝在排序中更为靠前；同时对于一些没有出现行为的、新鲜上架的宝贝，根据它的 side-information 对它进行 embedding 表达，有效解决宝贝的冷启动问题。

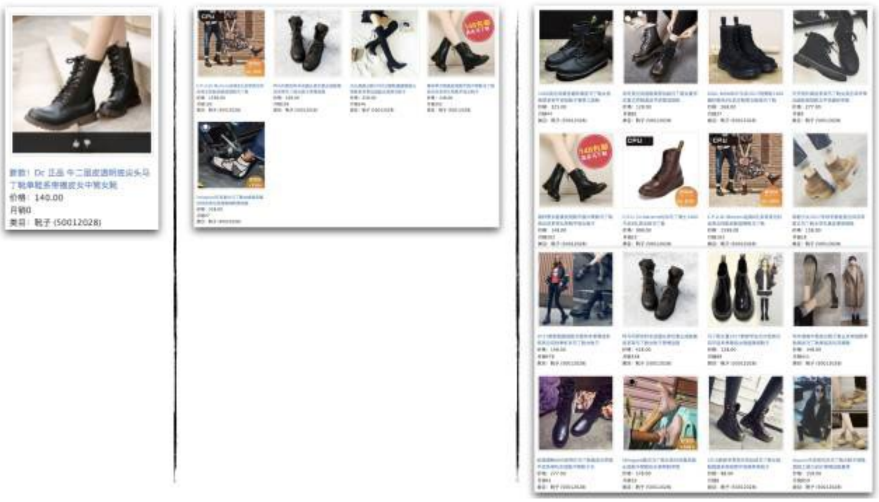
（3）S^3 最终版本

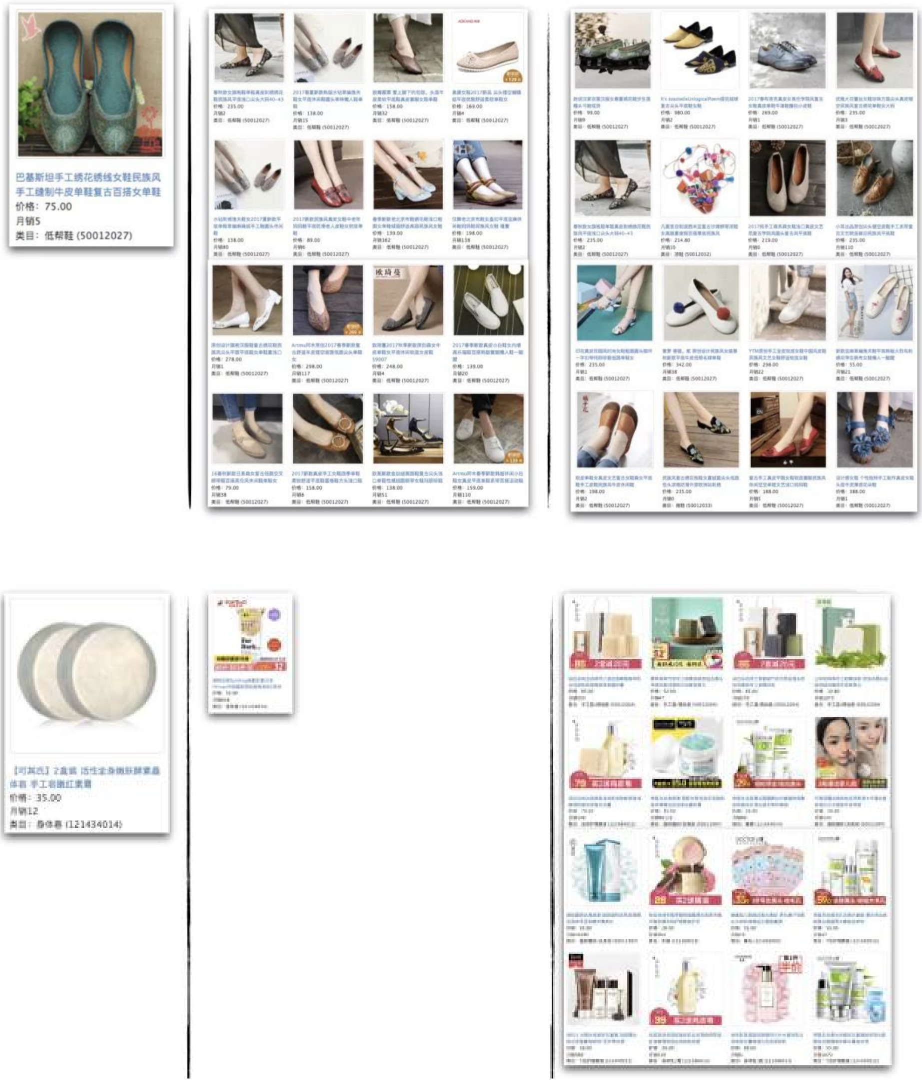
在引入 side-information 和 sequence 行为序列之后，模型准确性大幅提升，但仍然面对全网宝贝 embedding 参数空间太大、训练样本过多的难题（均在千亿量级以上）。为了解决这一问题，我们根据行为序列天然的转移概率重构全网宝贝的 weighted directed Graph，再将整个网络切分成多个 sub-graph，在每个 sub-graph 内部进行 Graph Embedding 训练，不同的 sub-graph 间并行训练，缩短训练迭代周期，网络结构如下图所示：



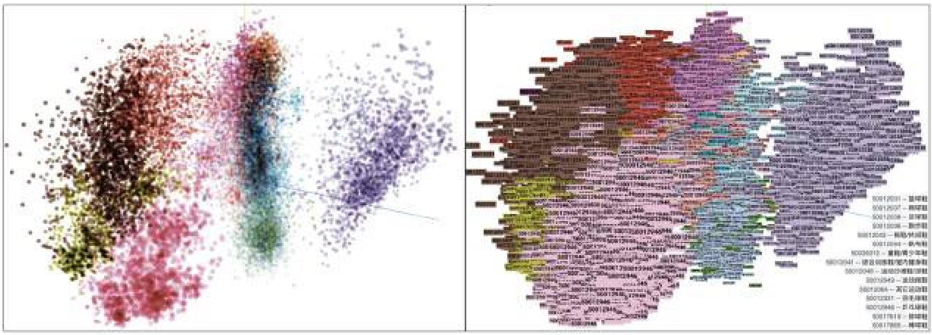
在最终计算 i2i 的召回环节，对 embedding 结果构建查询索引，基于 GPU 集群以 batch 形式进行高效的 Nearest Inner Product Search 最近邻检索，同时将整套框架落地在搜索工程团队的 BE 引擎当中，实现用户到 trigger 到 embedding 宝贝结果的实时召回。

相比于经典的基于共现的 i2i 算法，最终版本整体在召回上更加丰富，badcase 出现情况较少。全网商品在首图商品池召回的 case 如下，图中三列分别是原始宝贝、swing 召回、graph embedding 召回：





下面这幅炫酷图对我们的 embedding 高维向量有一个更直观的解释，我们随机选取了运动鞋这个一级类目下的宝贝 embedding 向量降维，不同的颜色代表不同的叶子类目，每一个点代表一个商品降维后的坐标，可以清晰的看出同一个类目下的宝贝 embedding 向量很 " 自觉 " 的 cluster 在一起，说明这种自觉的本质就是 graph embedding 的向量是靠谱的。

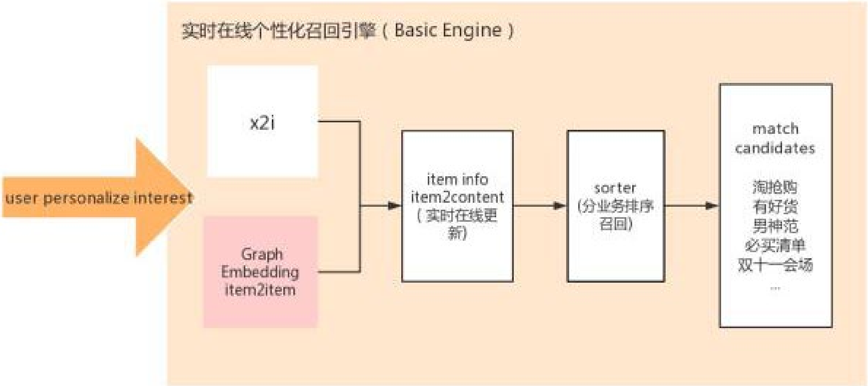


在运动鞋1级类目下商品Embedding可视化(PCA降维)
每个节点是代表一个商品，每种颜色代表一个叶子类目

1.3 在推荐场景中落地及调优

在算法的深度模型训练部分，我们依赖了搜索工程团队的 Porsche blink 系统，在 XTensorflow 上进行了模型 TF 版本的开发，构建全网序列行为图并完成采样，样本级别达到千亿，基于并行的 GPU 集群训练。整个召回框架中采用 subgraph 的拆图结构，也是加速迭代效率的关键点，同时在 PS 的参数大规模上涨的情况下，样本的遍历及样本的训练覆盖率对整个模型的收敛效果也至关重要。

首页个性化推荐框架在在线召回阶段，主要应用了搜索工程团队的 Basic Engine 在线化召回引擎，我们将 Graph Embedding 训练完成的模型用近邻检索的方式把数据载入到引擎中，且将原有的 offline,online 结合的方式全部在线化，增强了推荐系统的灵活性，实时性及召回能力，整体框架如下图所示。

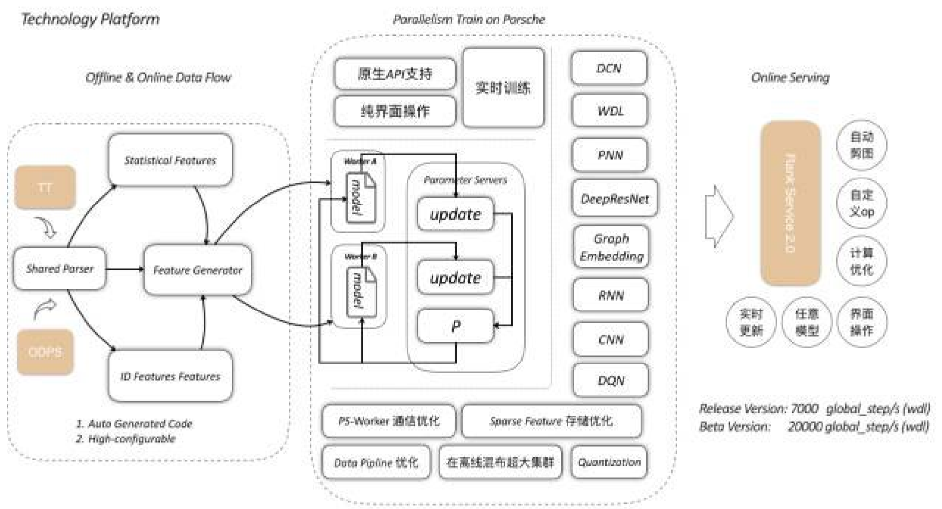


在双 11 之前，我们在这块的工作迭代调优过程历时近两个月，期间分别尝试了多种不同的网络模型结构，样本选取过滤方式，引入 sequential 行为信息、转移概率边的全新构建方法、side information 等多方面，多角度调优方法，在日常首页个性化（AIO）日常场景中 uctr 对比 swing 版本 i2i 提升明显，同时双 11 在该场景全面替换线上的最优召回方式。

2. 基于 XTF 的深度排序模型

2.1 XTensorflow 简介

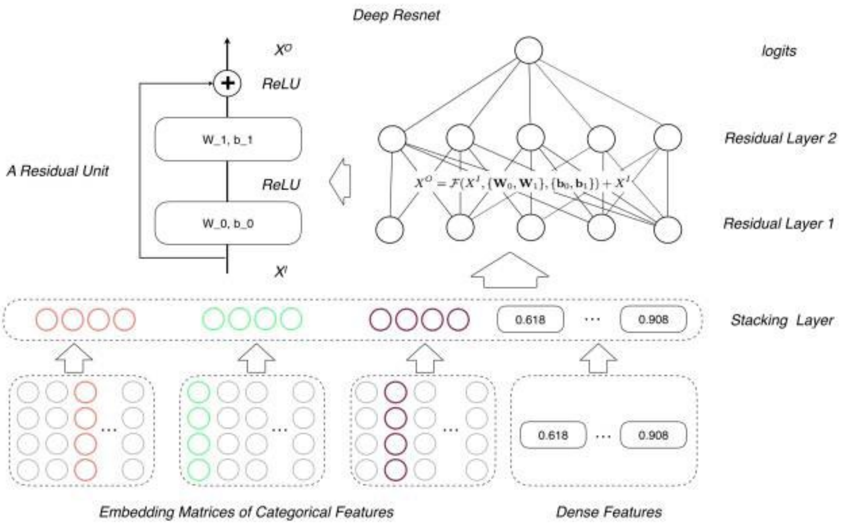
2017 双十一我们承接了手淘首页流量分发最大的几个首页场景，这几个场景的特点是流量大、业务规则限制多、同时业务变动频繁，我们需要一个稳定、能支持快速迭代和实时计算的机器学习平台来支撑我们训练模型以及在线打分。为此，我们参与了工程团队的共建工作基于 Porsche blink 的分布式 Tensorflow 训练及在线打分平台，首页算法同学基于该平台开发出了若干 Rank 模型，在相应的业务场景拿到了不错的效果，我们称该平台为 XTensorflow，简称 XTF。



在这个深度学习平台上，我们在双十一上线了包括 DCN、DeepResNet 的深层高阶特征学习模型，以及更加推广及成熟应用了 WDL 深度模型，相对于 WDL 模型，更为复杂深度模型的尝试也取得了明显的效果。

2.2 DeepResNet 在 AIOplus 场景的应用

在深度学习的推荐领域，当用户宝贝数据以及相应的参数膨胀到一定规模后，加深网络的深度增强模型的学习泛化能力是众所周知的方式，但网络盲目的加深又会同时引入参数爆炸、梯度消失、甚至过拟合等问题。参考 Resnet 网络技术在图像领域的获得的成功，其解决的根本问题就是当网络深度不断加深之后，梯度消失越来越明显，效果越来越差。关于这一点，之前我们在 WDL 的 deep 侧做过相应的尝试，随着网络导数及隐层结点数的增加，普通 NN 网络慢慢的训练效果会越来越差，甚至导致效果出现较明显的下降。基于这些实验现象，我们将 WDL 进行了 Deep Resnet 的深层扩展，基本原理图如下：



原始的 input 层包含实值特征及 id 类特征的 embedding 向量，接入一个 10 层的 Resnet 层，最后通过 logloss 来定义最终的 loss。同时，基于场景双十一用户行为变化迅速，以及 AIOplus 区块活动素材的多变性（运营会根据 BI 数据，实时调整素材等），训练了实时的 Deep Resnet Model，在预期热及双 11 当天在场景卡片会场业务上上线拿到效果。

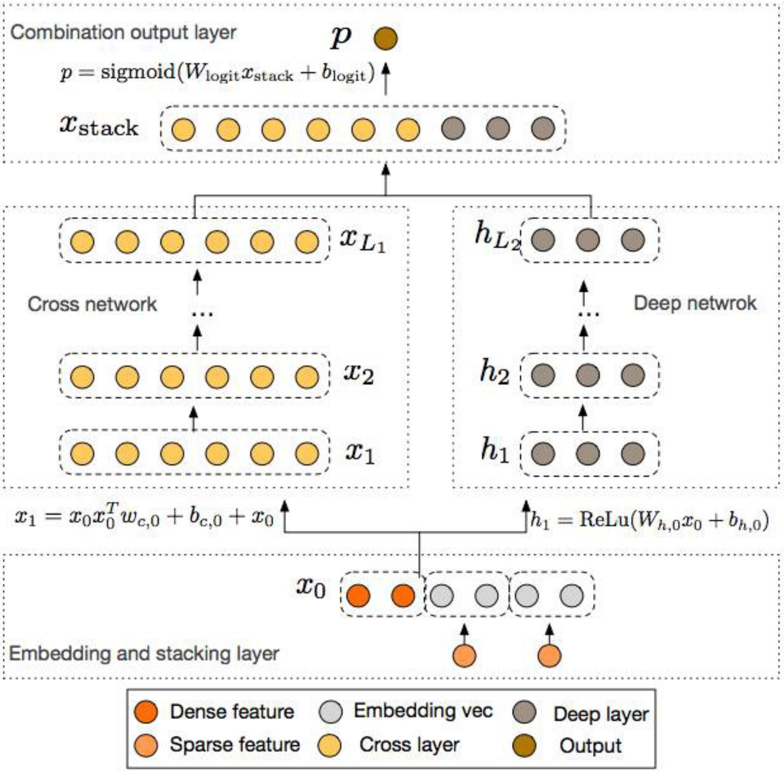
2.3 DCN（Deep & Cross Network）在主会场入口个性化的应用

通过将稀疏特征 embedding，配合多层全连接 Relu 层，DNN 能够自动进行特

征交叉，学习到高阶非线性特征，但是这种交叉是隐性的，交叉阶数无法显示控制，而且容易 "over-generalization"。为了弥补这一点，WDL 模型引入了 wide 侧，显示记忆一些可解释的特征交叉信息以保证模型的精准性。但是 wide 侧的引入带来了大量的特征工程的工作，同时也只能学习 2 阶（cross-product）浅层的交叉。DCN 引入了 cross network，通过网络层数控制特征交叉的阶数，并且它实现高阶特征交叉需要的参数量远远小于 DNN；cross layer 的定义比较简单：

$$\mathbf{x}_{l+1} = \mathbf{x}_0 \mathbf{x}_l^T \mathbf{w}_l + \mathbf{b}_l + \mathbf{x}_l = f(\mathbf{x}_l, \mathbf{w}_l, \mathbf{b}_l) + \mathbf{x}_l$$

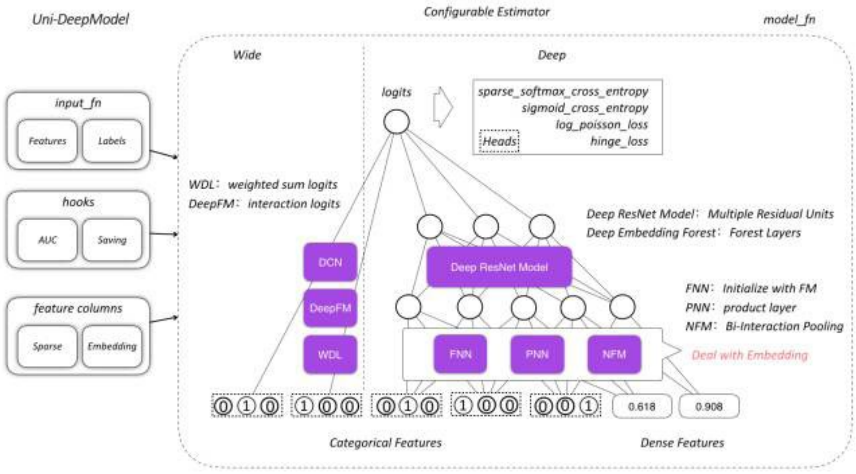
这个设计跟 ResNet 比较像，l+1 层的映射函数 f 实际上是在拟合残差 $\mathbf{x}_{l+1} - \mathbf{x}_l$；cross network 最终会和 deep network 进行 jointly learning，完整的网络结构如下图：



双十一主会场入口场景展位少召回相对充分对 top n 的精准度要求很高，因此我们选择 DCN 模型，借用 cross network 的高阶特征交叉提升 ctr 预估精准性，同时由于从入口点进去后主会场承接模块变化频繁，我们必须选择 item 粒度（而非素材 content 粒度）来建模，并且对模型泛化能力的要求较高，因此我们用了较深的 DNN 网络来提升模型的泛化能力。作为主要引流场景之一，我们有充分的数据来学习模型，最终上线的版本中，我们设置了 3 层 cross layer，以及 10 层 Deep Layer（Resnet 结构）。为了适应场景的变化我们做了大量的日志清理工作，相对于纯实时样本流，小时级样本流更加方便做复杂的样本清理，因此在调优阶段选择的是 incremental 的小时级训练，再切换到实时模型的方式。

2.4 Union–DeepModel

Google 提出 Wide&Deep model 之后，这种基于 Deep 侧做高维特征提取（泛化），Wide 侧进行显示特征交叉（记忆），最后进行 Jointly Learning 的框架被进行了各种各样的升级。我们整理并实现了里面比较经典的有意思几个模型，包括 WDL、PNN、DeepFM、NeuralFM、DCN、DeepResNet 等，这些模型均基于 TF Estimator 框架实现，封装为 model_fn，特征处理以及模型训练过程高度可配置。同时，基于这个长期在深度模型上的积累，提出了 Union 结构的 DeepModel，用以适应各种各样业务场景对深度模型的需求。



三、欢迎与我们交流

阿里巴巴推荐算法团队目前主要负责阿里电商平台（包括淘宝、天猫、海外版淘宝、Lazada 等）的商品及 feeds 流推荐，其中手机淘宝首图个性化、猜你喜欢、购买链路等场景每天服务数亿用户，涉及流量效率提升、用户体验、提高商家及达人参与淘宝的积极性，优化商业生态运行机制。

欢迎热爱算法，对业务有好奇心，有合作精神的同学一起工作、成长。简历可投邮箱：

binqiang.zhao@alibaba-inc.com

pipei.hpp@alibaba-inc.com

四、引用

[1] DeepWalk: online learning of social representations. Bryan Perozzi, Rami Al-Rfou, Steven Skiena. Proceedings of the 20th ACM SIGKDD international conference on Knowledge discovery and data mining,2014.

[2] node2vec: Scalable Feature Learning for Networks. A. Grover, J. Leskovec. ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD), 2016.

[3] LINE: Large-scale Information Network Embedding. Jian Tang, Meng Qu, Mingzhe Wang, Ming Zhang, Jun Yan, Qiaozhu Mei. Proceedings of the 24th International Conference on World Wide Web, 2015.

[4] entity2rec: Learning User-Item Relatedness from Knowledge Graphs for Top-N Item Recommendation. Enrico Palumbo, Giuseppe Rizzo, Raphaël Troncy. Proceedings of the Eleventh ACM Conference on Recommender Systems, 2017.

[5] Discriminative Embeddings of Latent Variable Models for Structured Data, H. Dai, B. Dai and L. Song. International Conference on Machine Learning (ICML). 2016.

[6] Deep Coevolutionary Network: Embedding User and Item Features for Recommendation. H Dai, Y Wang, R Trivedi, L Song. Recsys Workshop on Deep Learning for Recommendation Systems. 2017.

[7] Predictive Collaborative Filtering with Side Information.Feipeng Zhao and Min Xiao and Yuhong Guo. International Joint Conference on Artificial Intelligence(IJCAI),2016.

[8] ICE: Item Concept Embedding via Textual Information. Chuan-Ju Wang, Ting-Hsiang Wang, Hsiu-Wei Yang, Bo-Sin Chang, Ming-Feng Tsai. Proceedings of the 40th International ACM SIGIR Conference on Research and Development in Information Retrieval, 2017.

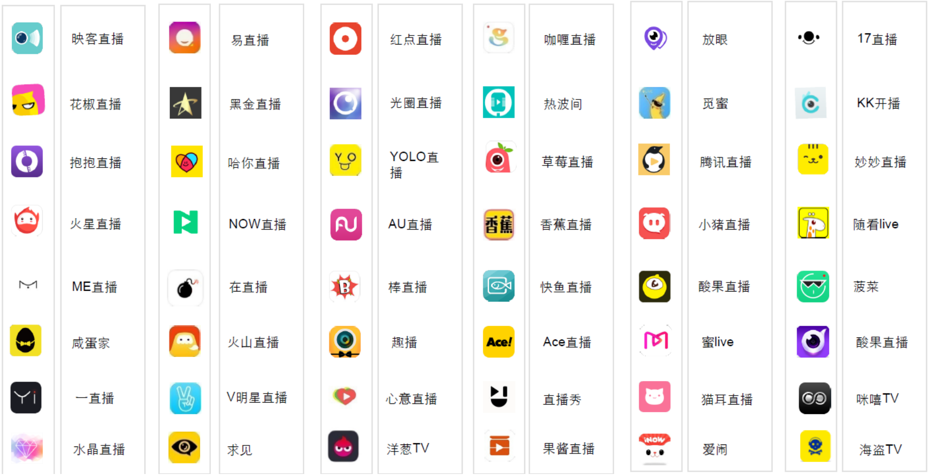
[9] Distributed representations of words and phrases and their compositionality. Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg Corrado, Jeffrey Dean. Proceedings of the 26th International Conference on Neural Information Processing Systems, 2013.

[10] Meta-Graph Based Recommendation Fusion over Heterogeneous Information Networks. Huan Zhao,Quanming Yao,Jianda Li,Yangqiu Song,Dik Lun Lee. Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining,2017.

双十一安全技术：目标检测在淘宝直播中的应用

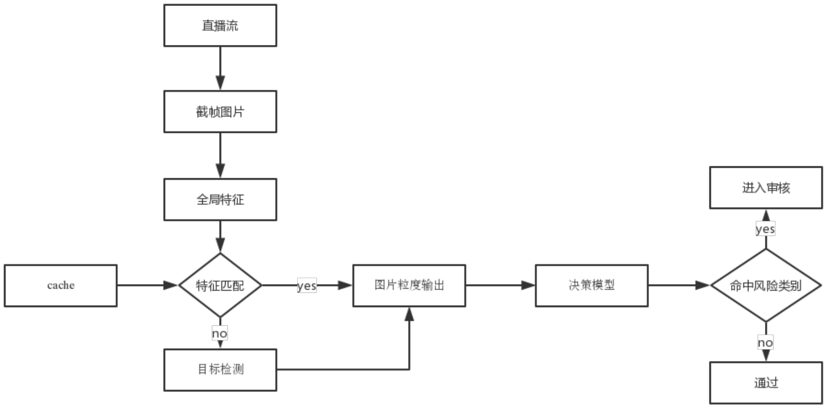
炫谦

背景



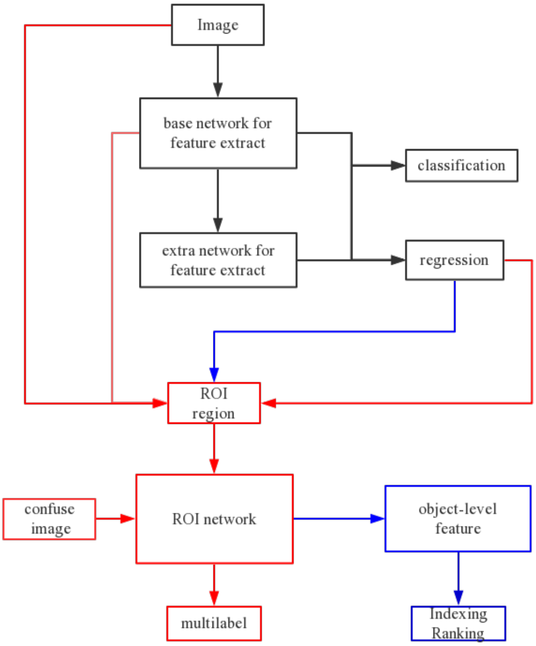
- 2016 年为直播行业的元年，行业井喷式发展，各种直播平台涌现
- 2017 年直播行业进入缓和期，但淘宝直播仍然保持着增长，业务量的增长伴随着风险量的增加
- 在风险类型方面，占比量较大的着装要求方面已经使用黄图识别模型进行覆盖（淘宝直播尺度非常严格，“着装不正”并非露点和大尺度的低俗，而是一些领口较低、夏季的半透视装等）
- 风险占比最高的版权（翻拍）风险仍然暴露在外，依靠人力进行防控，这部分风险也是上线模型的主要防控风险点

目标检测整体技术方案

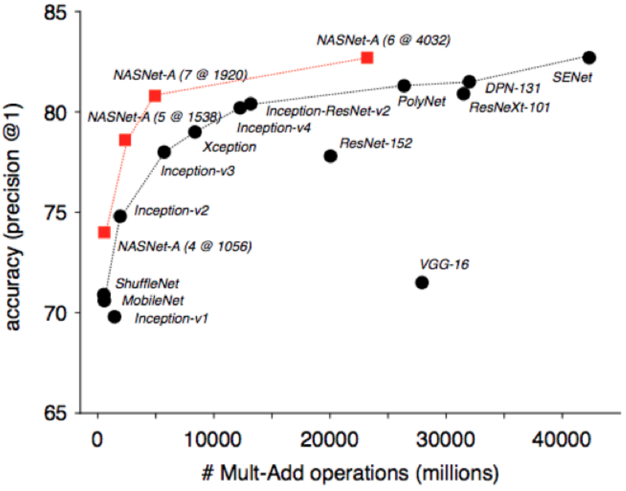


- 适应直播业务的快相应，高并发的需求，增加图片全局特征去重模块对同一直播流的时间序列上的截帧进行特征匹配去重，减少后端复杂模型的调度量，算法处理耗时在 ms 级别

目标检测网络

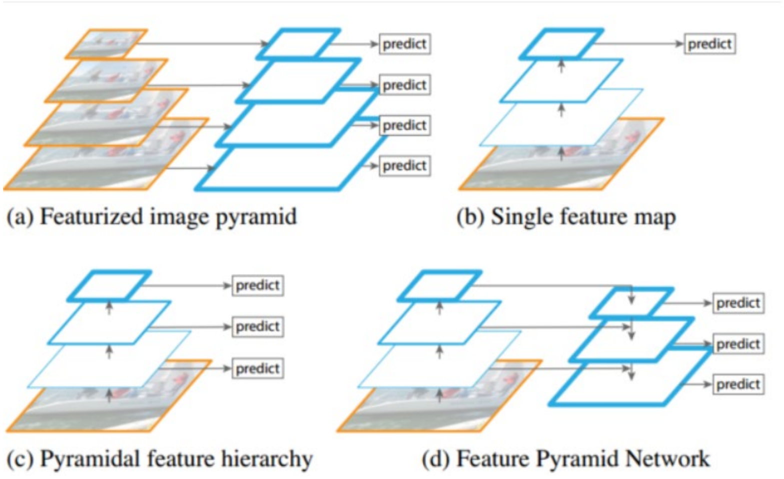


1. base network 对整图的特征进行提取



目标检测 base network 的特征提取结构与 imagenet 的图片分类任务 CNN 网络相似，作为整图特征抽取网络比较常用的为 VGG（baseline）、ResNet 系列（特征具备高表征能力）、Moblienet（小模型），业务应用模型其实是模型的准确率和效率的 trade-offs 问题

2. extra network 适配目标多尺度



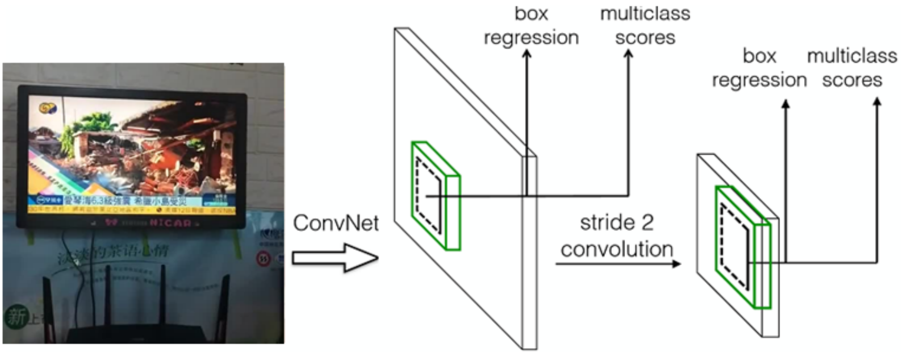
（a）的多尺度方式应用于传统特征抽取方法，比如基于（SIFT、SURF 等）特征的目标检测方法，现有方法由于特征抽取代价太大，对于原图的多尺度方案基本不再使用；

（b）利用 CNN 网络结构的图片表征抽象能力，最后一层的特征具备最好的语义特性且具备尺度泛化能力，但是对于最后一层具有较大的视觉感受野，pixel-level 的表征能力有限，这一点对于定位任务会带来不利因素；

（c）可以简单理解成将（a）中的 downsample 操作转移到了卷积层，同时利用高层的语义特征和低层的像素表征特征；

（d）在（c）的基础上添加特征层之间的 context 信息，最终达到特征的更准确表征

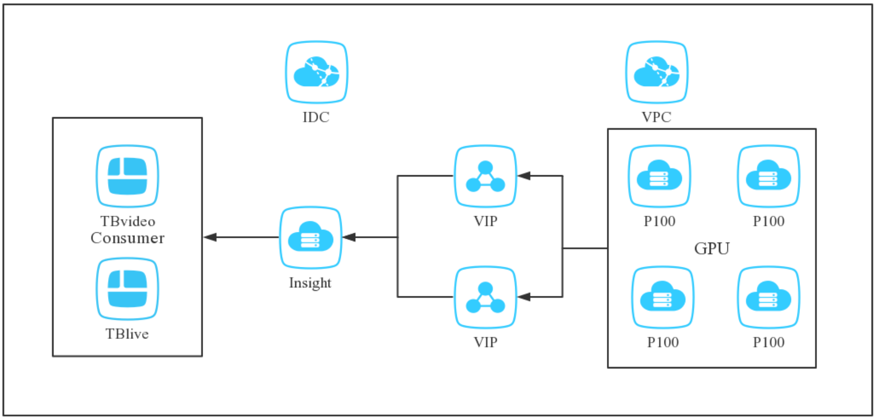
3. regression 对翻拍内容进行外框定位

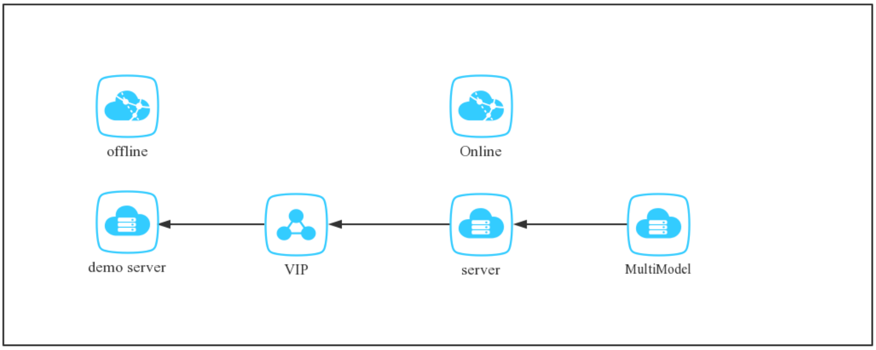


4. ROI network 对翻拍内容进行细粒度识别，同时能够对易混淆样本进行约束

5. Demo 展示效果：

服务部署





1. Docker 化部署云上 VPC 环境（降低扩容时间成本）
2. VIP 进行分组隔离，保证双十一集群稳定服务
3. 跨域调度，利用云上便捷资源
4. 内部网络 Demo 跨域，多模型组合，调用线上服务

持续迭代下的双 11 供应链体系最新架构及功能解读

粤谦

阿里巴巴资深技术专家粤谦

粤谦，阿里巴巴资深技术专家，创建供应链算法团队，负责供应链平台事业部算法研发，通过综合使用多种算法来保障供应链的整体运行效率。

供应链概述

天猫双 11 是一年一度的盛事，大量的用户会在双 11 期间来到阿里巴巴的平台上购物，这也就意味着会产生海量的订单和包裹。这些包裹大小不一，分布在全国各地，且数量庞大。如何快速、高效的处理这些订单，将消费者的包裹尽可能快的送达

到消费者手中，成为双 11 期间一件非常重要而艰巨的工作，其背后依赖于整个供应链的高效协同。

阿里的供应链系统分为上下游两个部分，上游涉及业务决策层，包括商品、价格、营销活动、营销策略、供应链计划等模块，上游会将结果传输到供应链的下游模块，包括订单情况、库存管理、补货调拨的动作等，然后上下游协同起来构成了整个系统的架构体系。

从地理位置和仓网结构来看，供应链网络极其复杂，不仅有大量的仓，仓的角色还十分多样，比如有全国中心仓、区域仓、前置仓等，因此如何将上下游结合起来进行统一的决策是非常具有挑战性的问题。

在整个供应链中台体系中，供应链计划具有非常重要的意义。从宏观上来说，目标是希望建设一个协同、高效、经济的一个供应链体系，通过不断的循环、迭代、优化形成最优策略。首先是进行目标设定，通过算法计算得到决策的方向，然后通过实时数据的回流在结果上进行反馈，最后把结果与当初的目标进行比对，评估执行情况，如果未能达标就进入下一轮的循环，这就是供应链系统不断迭代和优化的过程。

此外在供应链建设中还有一个很重要的命题：供应链多模块协同。上游的模块包括有会员 CRM、商品管理、品类规划、品牌调性、选品、价格管理、营销管理、商家管理、流量管理等模块，下游的模块包括有 S&OP、预测、补货、调拨、库存、订单、履行、仓网结构、第三发配送等模块。这些复杂模块之间如何达到有机的联动并形成整体运作的高效体系？这是我们一直在研究的问题。比如在大促期间，如果某项业务需要在某个区域进行营销活动，并且制定了一系列营销的方案和计划，但是如果后端整个仓的产能和配送无法协同的话，就容易造成上游生产了很多订单、但下游无法配送出去的情况，会极大损害消费者的时效和消费体验。

2017 双 11 的挑战

2017 的双 11，给整个供应链系统提出了更高的要求：

- 准确的细颗粒度需求预测，需求预测是整个供应链的起点，要求支持后端 SKU 级别；

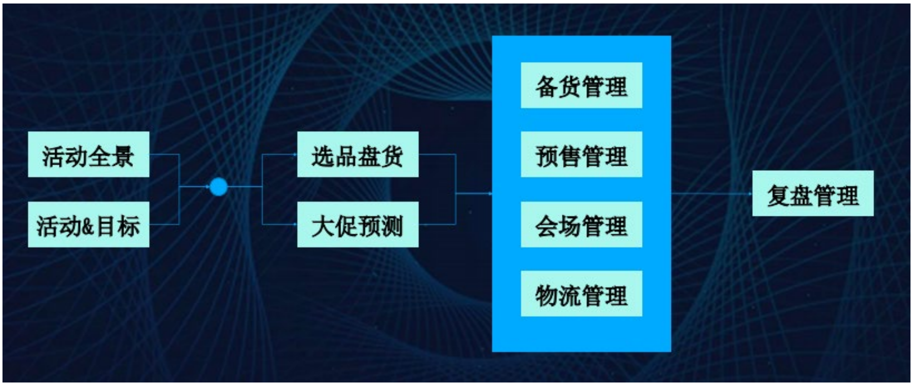
- 商品种类极多，各种优惠促销策略极其复杂；
- 提前进行大量备货入库，入库量非常庞大；
- 仓库库容、运配网络都有产能上限；
- 大量商品需要在广袤的地域上进行合理的库存分布；
- 海量订单的时效如何保障，这是双 11 条件下很重要的一点。

以上这些命题都是需要解决的问题。

预测备货

需求预测是整个供应链的起点，更遑论双 11 背景下提出的准确的细颗粒度需求预测，这涉及系统的算法研发层面，具体来看分为两方面的工作。第一方面，需要在数据层面上进行数据收集、清洗、特征准备的工作，这里的数据非常纷繁复杂，包括过往历史的用户行为数据、商品销售数据、以及双 11 期间具体的促销信息（包括参与促销的商品种类、促销策略、满减、满赠等促销信息）。第二方面，需要考虑算法模型层面，主要涉及三个算法：传统的时间序列算法、常用的机器学习算法以及双 11 期间研发完成的深度学习算法，通过反复的调试致力于将这三类算法高效地融合在一起，最终形成了一整套的算法模型解决方案。这两项工作结束之后就可以生成销售预测和各种颗粒度的预测，比如前端的商品、价格预测等细颗粒度，整个品牌、行业销售情况的粗颗粒度，还可以用于后期的产品企划。

在预测的基础之上要进行大促预热期的备货，整个双 11 的备货起点起始于 10 月中旬，但其实在这之前整个供应链的备货工作就已经展开了。为了辅助整个业务的便利性，系统进行了全局视图的项目，以期在多模块下实现跨模块的决策，各个阶段都能在统一视图进行管理，这也便于后期的复盘活动。

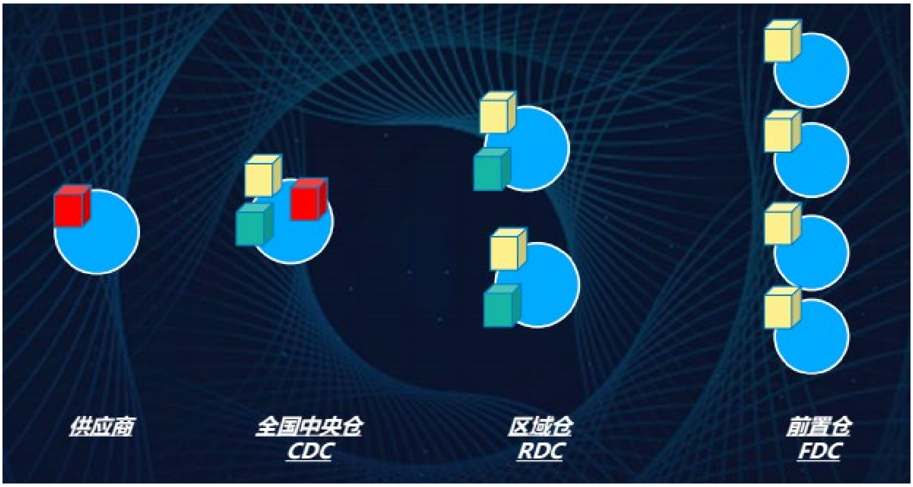


全局可视化

补货调拨

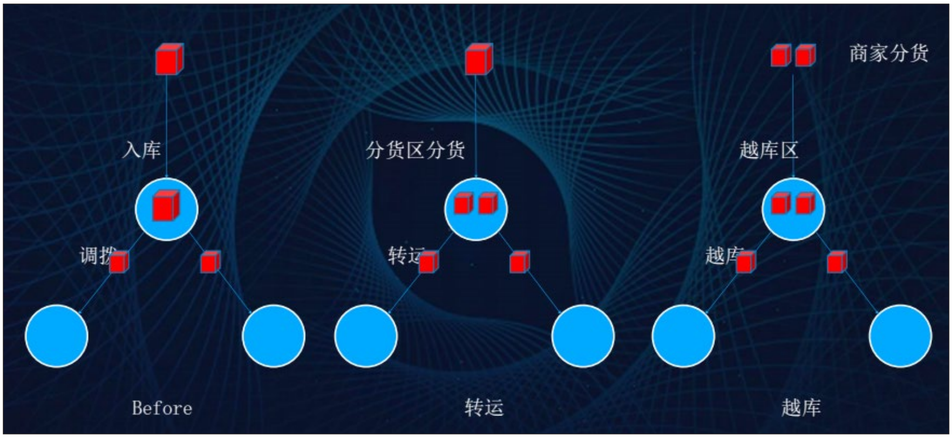
补货调拨也是库存领域非常重要的环节之一。在这一部分，需要进行动态库存布局、转运 / 越库策略、预售 / 爆款下沉等主要活动。

首先是动态库存布局。仓网结构复杂且角色各为不同，如果想要互相支援需要考虑好两个问题：如何把库存放在离消费者最近的地方？如何平衡存货成本和时效？因此系统的将仓的结构进行多层次级别的划分，爆款放在前置仓，其他放在 CDC 或区域仓，这背后的原理是根据销售速度再通过算法引擎来驱动。



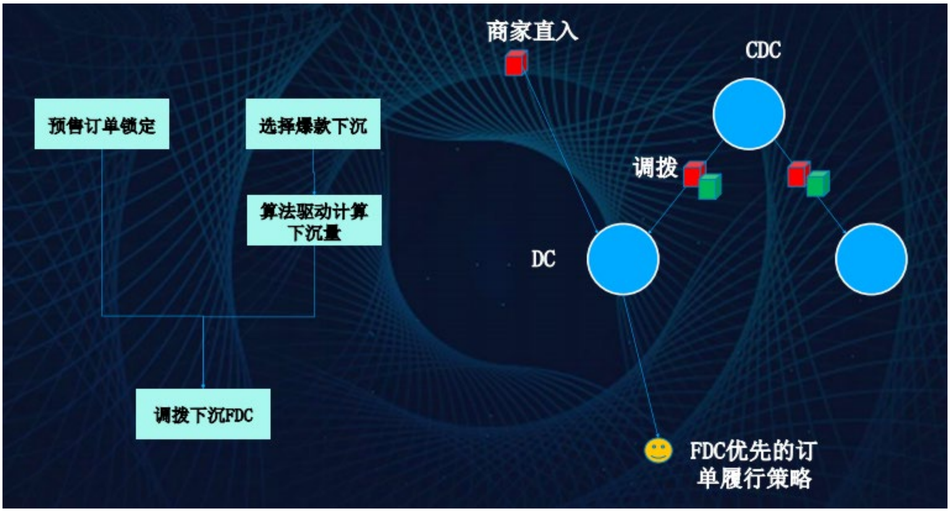
动态库存布局

此外是转运 / 越库策略，这在双 11 期间已经得到了实践应用，入库效率显著提升。之前的工作方式是将商品统一送到 CDC 仓并直接上架，然后通过调拨逻辑调拨到前置仓；转运是指商品分货区分货，这样可以减少成本并提升效率；越库则在于商家分货，因此处理速度更快。



转运 / 越库

还有一个重要的双 11 项目是预售 / 爆款下沉，其核心思想是将商品尽可能地放在靠近消费者的地方，能够缩短发货链路、缓解区域仓的压力，而且前置仓通过原箱发货、预打包等措施能够大大提升发货效率。预售情况下，商品订单已知可以提前备货；爆款商品则会提前考察，考察维度除了销售速度还有发货速度等，后续就可以有序调拨到 DC 仓，能够快速拣货打包。



预售 / 爆款下沉

仓配履行

仓配履行环节涉及预约配送、主动截单、订单生成调控、智能仓内操作、运配计划等动作。

预约配送是为了缓解物流压力，联动上下游在时间上对订单进行错位，在上游激励消费者选取预约配送，缓解压力的同时也能够保障时效性。

主动截单与整个仓容挂钩，需要在订单履行系统里进行截单操作，视仓库处理能力的上限来定，一旦接近上限就需要进行截单、蓄洪、然后等到压力缓解之后再下发订单，这样可以保证整个仓的生产有序进行，避免瘫痪且保护仓的产能。

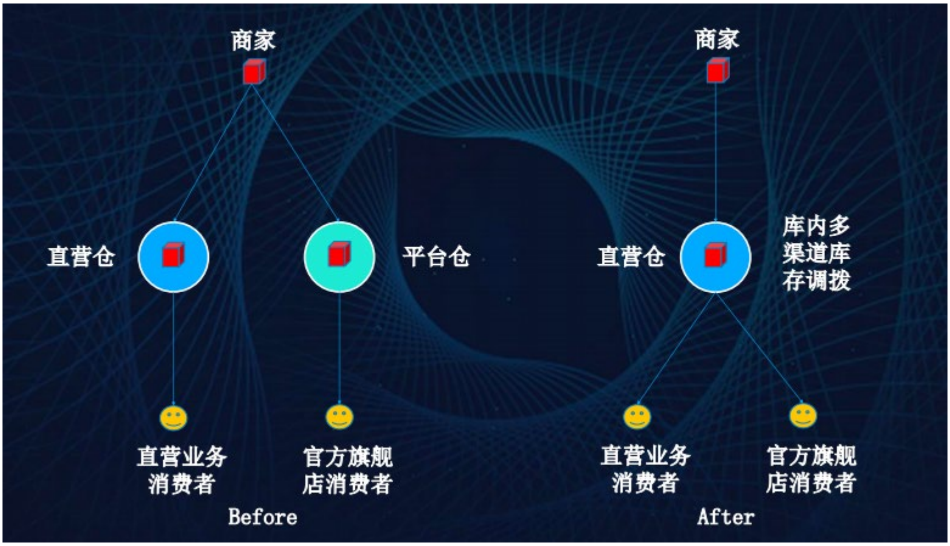
订单生成调控同样以保护仓的产能为目标，这是通过控制订单生成方式来控制的，一般会采取两个方案：流量调控和营销调控，如弱化营销力度、削弱订单生成力度，以形成有序的订单生成过程，其背后的逻辑都是为了缓解仓的压力，避免爆仓。

智能仓内操作是指在仓内使用智能机器人 AGV，基于一些智能化的操作来提升拣货效率。仓储往往会面临着货品摆放、拣选路径、自动化、人效提升等挑战，而机器人本身可以自我学习从而不断优化，此次双 11 的实践效果也表明拣货效率至少提升了三倍有余。

仓后的运配主要面临的问题是如何在满足运输要求的前提下，对现有资源实现最大化的利用。运配计划的制定和执行可以有效地削断运配路径和成本，比如在不违背单车运能、时间等相应的限制因素的前提条件，对多订单寻求最大程度的拼载策略；在出库链路，按照规模化运输的策略，通过一级 / 二级分拔 / 集货来执行干线运输等等。

商家协同

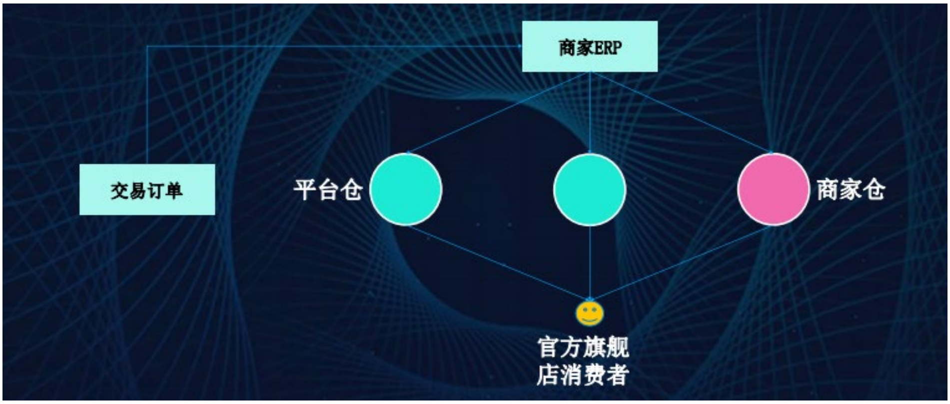
在商家端跟商家协同同样是十分重要的工作，主要包括一盘货管理、ERP 订单协同 & 商家仓协同。首先是一盘货管理，之前的工作模式下经常会带来商家两地送货、成本较高等问题，双 11 融合了自营业务和平台官方旗舰店业务的供应链体系，商家只需一次送货就可以满足多售卖渠道的需求，降低了商家成本。比如雀巢的当日达、次日达占比从 30% 提升到了 70%，跨仓发货比从 60% 降低到 10%，运货成本和操作成本都大大降低。



一盘货管理

此外是 ERP 订单协同 & 商家仓协同系统，赋予商家更加方便快捷的订单履行决策。对于商家来说提升了决策效率，对于整个供应链体系来看多了一个商家仓的角

色，所以可以将商家仓的角色也纳入到仓网体系中去，有助于统一协调外部资源，能达到更好的管理效果。



ERP 订单协同 & 商家仓协同

未来展望

今年的双 11 创造了销售奇迹，双 11 结束后至今整个项目组也一直在进行复盘活动，包括产品化思考、内容沉淀和反思：

- 2018 年供应链的系统模块会更复杂化，仓网结构更复杂化。在复杂度增加的情况下，应该如何更好地去平衡需求与供应的矛盾是项目一直在研究的难题；
- 大的潮流趋势之下，需要思考如何通过大数据、IoT、社交网络等新技术实现供应链 + 的结合和升级；
- 个人经验和智能学习的碰撞。智能化算法不断提出的背景下，强化学习、深度学习等新型决策机制应该与个人经验形成有机的、智能化的解决方案；
- 供应链对实时反馈和智能化的要求不断提升，所以要更好更快地提升整体效率。

七层流量清洗提供安全防护新方案

铁花

铁花，阿里巴巴资深技术专家，淘宝最早 SDL 建立及实施人，
淘宝第一代 web 安全解决方案主要开发，安全静态代码扫描平台创建者。

霸下，龙之九子之一，因其喜负重与底层系统之意契合，遂取此名。

基于场景清洗恶意流量

传统的恶意流量来源于攻击、扫描等，例如 DDOS 攻击、CSRF 攻击、XSS 攻击，或使用扫描引擎如 APPScan 对网站的漏洞尝试扫描。与之对应的有传统的抵御方式，如云平台上针对 DDOS 的防御，用 WAF 平台抵挡 Web 的漏洞扫描。但是，双 11 存在大量基于场景的攻击，这些技术手段并不能达到我们的要求。

例如，在下单场景中，用户经常抱怨在秒杀活动开始后还没来得及操作，活动就已经结束。这是因为背后有恶意集团进行有组织自动化地下单来抢购有差价的商品。我们把这些定义为场景化的恶意流量攻击。霸下 – 七层流量清洗平台正是基于场景清洗恶意流量。

如上图所示，霸下将左边混杂有恶意攻击的流量清洗，将纯净流量提供给业务系统，从而减少其在处理恶意流量过程中造成的带宽和计算资源的浪费。

提供突出防护能力

DDOS 攻击防护、CC 攻击防护、Web 攻击防护是所有平台提供的防护能力。霸下在此基础上更有效地抵御了“薅羊毛”的行为。“薅羊毛”行为包括批量机器下单，如在商品首发时争夺商品，套取积分获取最大利益等，极大地影响了用户体验。以上流量是机器行为，不是我们期望的用户流量。

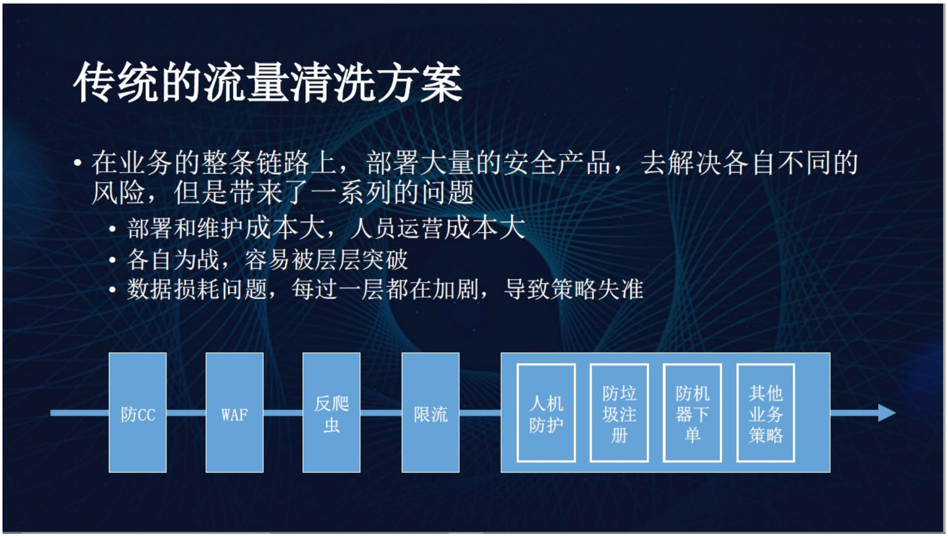
目前黑灰产企业的攻击更趋向于基于场景的、复杂化的攻击链路，区别于之前单纯的攻击方式。场景化的攻击包括但不限于：

- 竞争厂商使用爬虫程序爬取对方商品价格，在此基础上降价以获取优势
- 黑灰产企业恶意修改搜索排名加权条件，影响搜索结果
- 使用机器行为批量抢红包下单进而变现获取利益

霸下平台提供优秀的场景化攻击防护能力，有效地抵御了以上几类恶意攻击。

此外，霸下提供额外的网络限流功能。当流量超过网络的能力值时，我们对流量进行有选择的限流。值得注意的是限流在清洗之后，这样使限流的对象变为纯净流量，极大地提升了工作效率。

霸下 VS 传统清洗方案



传统清洗方案对于不同的攻击都有单独的防护模块，如针对 CC 攻击、DDOS 攻击都部署特定的模块。这样的架构集中在了业务上，导致系统在计算、带宽资源消耗了后才能发挥防护作用。传统清洗方案存在如下弊端：

- 每一个系统都需要部署和运维，不同运营人员之间沟通不畅可能导致前后防护对象的重叠，更有甚者前后规则的冲突可能引发系统故障
- 模块之间缺乏协作，各自为战，可能出现攻击突破每一层防线，却无法问责的情况
- 数据消耗。每个模块全流量计算产生大量消耗，每层贴标签加入字段扩大了数据量，甚至可能产生数据篡改，导致最后策略不准

针对如上问题，介绍霸下 – 七层流量清洗理念。

1. 快速接入，统一接入

传统接入需要了解应用架构，统一还是特殊的接入层、应用上是否有连接层，应用上采用何种协议等因素都会影响接入速率。特别是遇到紧急情况，例如当遭受爬虫攻击导致系统濒临崩溃时，部署测试需要大量时间，无法起到应急作用。霸下实现了快速接入，统一接入，这得益于系统完整统一的架构，用户接入时只需要跟运营简单配合或者经过 BSOP 控制台简单操作就可以打通。

2. 纵深防御，数据打通

沿着客户端 –> 网络层 –> 应用层 –> 内部的流向将数据全部打通，所有计算指标和数据都是共享的，达到一份流量 copy 解决多份问题。数据打通给后端带来最精准的模型识别方式，提高识别准确率，同时也不需要浪费大量时间和空间的数据。

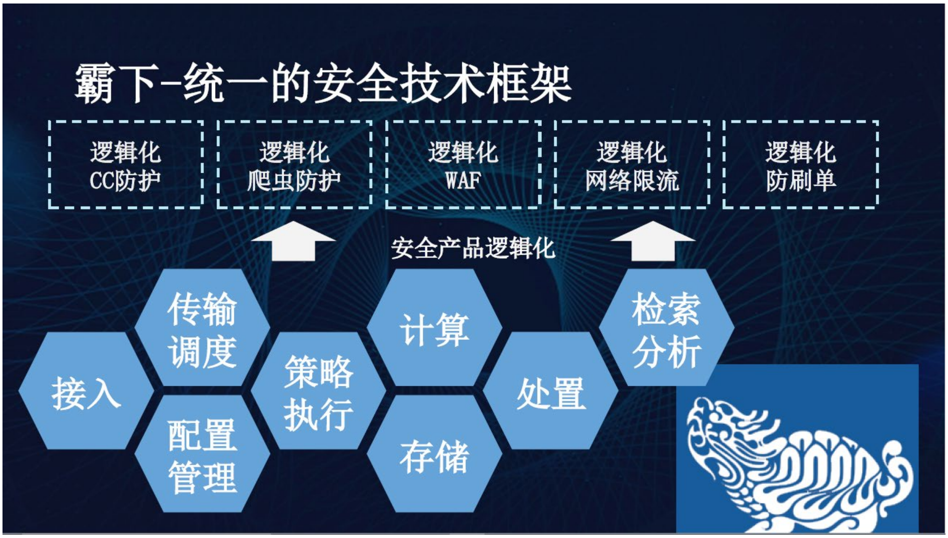
3. 自动升级，主动防御

以 DDOS 攻击为例，当攻击发生时人为的抵御无法对抗机器攻击，这需要我们通过策略的拦截效果实时监控，可以动态拉起、降级、收紧、放松策略而不需要人工参与，完成自动、主动防御。

4. 机器学习，策略推荐

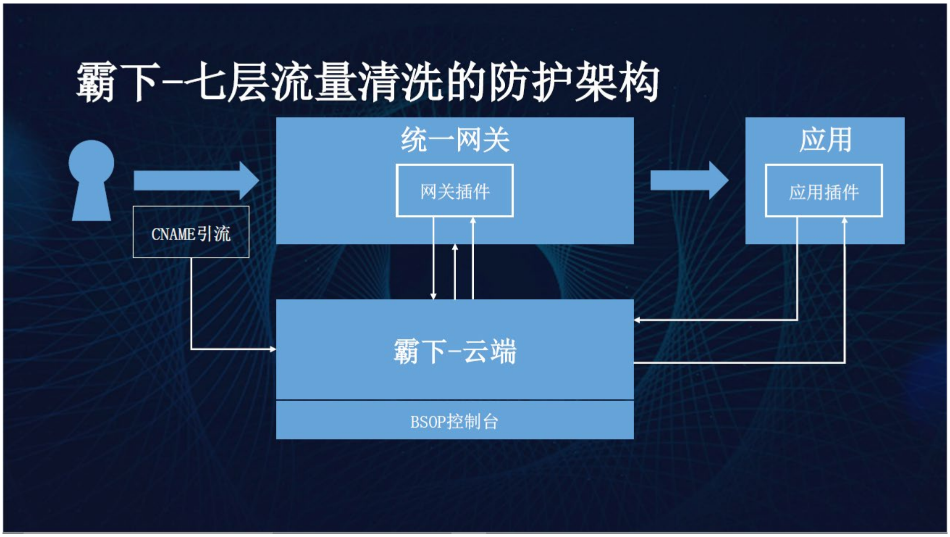
使用纯粹的规则无法预测下一次变异攻击，另外 CC 攻击、限流等人为防御会有偏差。霸下可以根据正常量告诉推荐值，自动生成策略、模型和对应的阈值，有问题时实现自动上线。

统一的安全技术框架



传统防护模块各自为战，均有自己的接入方式、传输调度、配置下发管理方式以及策略中心。策略中心涉及原始流量、指标，包括计算、存储、处置和检索分析的过程。霸下打通这种独立的格局，并且出于对用户习惯的尊重，在系统统一的基础上逻辑分化出不同防护模块。

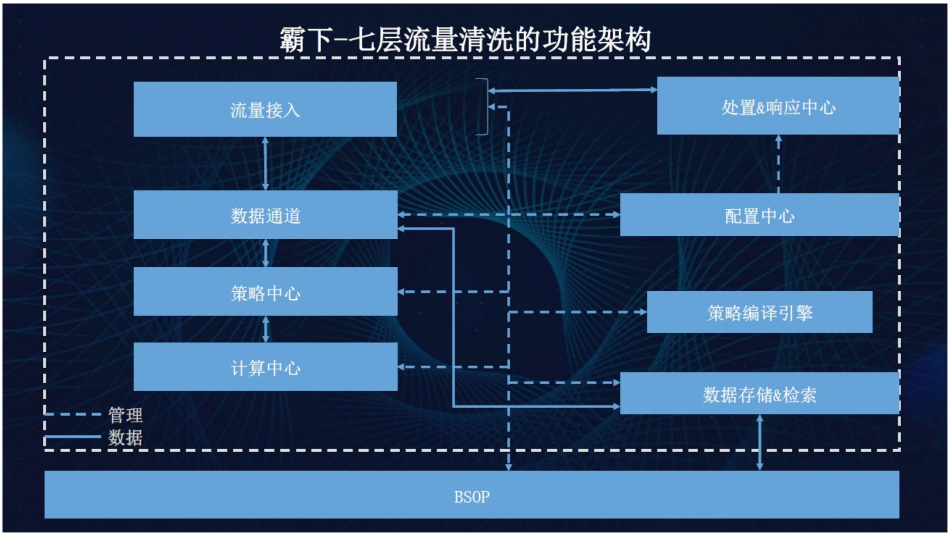
霸下的防护架构



移动互联网使用统一网关保持长连接、心跳等。用户访问某个应用的信息时，先跟统一网关建立连接，由网关进行路由。以下介绍霸下几种部署方式：

1. 统一网关上部署，实现全流量接入，快速策略部署下发。
2. 应用上部署插件，应用之间直接交互，但会多一些网关的消耗。
3. CNAME 方式。Web 应用不喜欢被侵入，数据通过 CNAME 引流后经霸下清洗返回到统一网关。
4. 结合的方式。统一网关和应用均部署插件，统一网关筛选全部流量，应用监控独享流量，保证没有流量遗漏，或者实现不同地方部署不同策略的要求。

从功能上看架构：



如上图虚线为管理模块，实线为数据流转模块。流量接入通过数据通道下发到策略中心，策略中心根据不同要求下发到计算中心，计算中心进行在线的实时计算返还给策略中心，有问题则做出处置相应，下发给端。此外，配置中心处置配置问题，编译引擎完成策略编译，数据检索用于查看误拦截或分析策略正确与否。最后，所有交互通过 BSOP 这一后端控制台完成。

霸下 – 七层流量清洗特点

1. 分布式计算 / 防护架构

清洗更快，清洗能力和部署机器数量呈线性增长，能够应对超大规模的安全防护。

2. 云端多维度、多方法的指标计算

防护 CC 攻击是一个维度，计算一个 IP 一秒内访问多少次作为指标。另外，从用户的角度出发，一个用户访问接口的频率是另一个维度。一个维度时采用计算的方式代价最小，效率最高，但问题在于不能支持复杂场景。现在，我们可以同时考虑 IP

和用户两个维度，实现应对复杂场景的需求，并且提供非常灵活的策略开发、灰度、线上观测机制。

3. 直观数据查询和拦截分析

通过流量清洗的工作台，非常直观的看到各个业务的安全情况，例如拦截、误拦截，并提供了诸如数据查询和拦截分析等运营能力。

4. 智能化

攻击导致崩溃后，通过监控自动启动拦截方式。智能化也是以后的发展方向。

案例

霸下 – 七层流量清洗负责阿里巴巴集团的所有网络层流量清洗和保障工作，这里主要提及两个数据：

1. 双 11 处理了峰值 2000 万 PV 的 QPS
2. 双 11 保障到达核心交易系统的流量纯净度大于 99.85%

2017 双 11：区块链在天猫国际商品溯源中的应用

游杭

背景

天猫国际正在全面启动全球溯源计划——将利用区块链技术、药监码技术以及大数据跟踪进口商品全链路，汇集生产、运输、通关、报检、第三方检验等信息，给每个跨境进口商品打上“身份证”。这项计划未来将覆盖全球 63 个国家和地区，3700 个品类，14500 个海外品牌。共同参与该计划的包括英美日韩澳新等多国政府、大使馆、行业协会以及众多海外大牌，中检集团、中国标准化研究院、跨境电子商务商品质量国家监测中心等“国家队”也已加入，通过定制天猫国际统一二维码并在码上合成全程监测手段，确保国内消费者买得放心。

2017 年被称作区块链应用的元年，区块链是一种为实现机构间和防止篡改的分布式记账系统，它在分布式共识算法、智能合约、加密算法等的基础上，可解决信任缺失场景下进行交易的问题。区块链的可追溯的特点可解决金融、征信、版权、证明等行业目前所存在的诸多痛点。将区块链技术应用于商品溯源中，可以提升商品整个流转过程中的透明度，对供应链形成更加全面有效的把控。

产品形态上，我们希望通过给进口商品打上唯一的身份证码，在双十一作为天猫国际业务的一大业务亮点，将商品整个生产、检测、运输、通关等环节的信息完整地展现在用户面前，提升用户购物体验，加强平台正品心智；同时创新地使用区块链技术，联合多外部合作方共同打造更具公信力的溯源平台。

项目简介

整个项目的价值主要体现在以下几个方面：

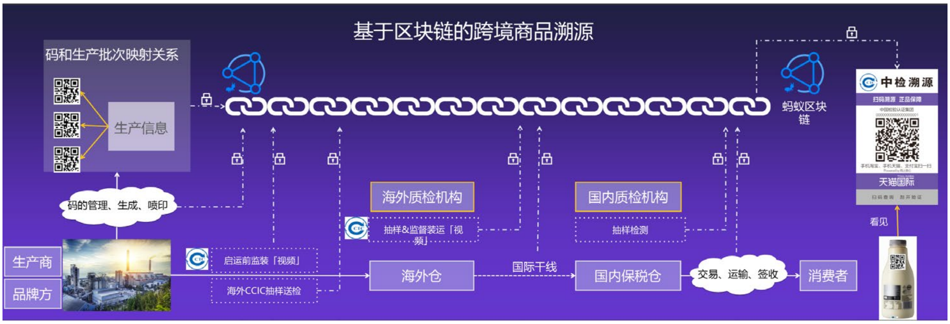
标准建设：溯源项目利用平台的商家、商品及供应链管理能力，建立全球商家 & 货品标准化档案，在 EWTP 框架内建立起一套跨境商品质检标准及全球质检机构网络。

货品把控：商品溯源可以和供应链中台进行很好地融合，从货品的生产到入仓的各个环节，都可以提供很好的底层数据支撑和货品质量把控。

正品保障：在消费者层面通过终端化的溯源二维码及公开透明的区块链技术支持，培养用户的正品心智，同时提升品牌价值。

商品溯源体系可以分为：生产企业溯源、海外商品溯源、国际物流及进口申报溯源、溯源信息终端查询四个方面，目前除生产企业溯源外其余体系都已初步建设完成。

商品溯源链路涉及生产企业、海外质检机构、物流企业、消费者四部分，需要各个部分通力合作，具体链路如下所示：



从上图可以看出，商品溯源的一个特点就是链路较长，并且线上线下相结合，既注重规则的建设，也注重仓库层面的实操。

区块链

简介

区块链技术，也被称为分布式账本技术，起源于比特币，核心优势是去中心化，能够通过运用数据加密、时间戳、分布式共识等手段，在分布式系统中实现基于去中心化信用的点对点交易、协同，从而为解决中心化机构普遍存在的高成本、低效率和数据存储信任等问题提供了解决方案。

特点

去中心化，区块链的颠覆性特点，它实现了一种点对点的直接交互，排除了被中心化代理控制的风险，使得整个交易流程更加透明真实。

公开透明可追溯，区块链中的每一笔交易都通过密码学方法与相邻两个区块串联，因此可以追溯到任何一笔交易的前世今生。系统中每一个节点都拥有最新的完整数据库拷贝，修改单节点数据无效。

稳定可靠，分布式的网络架构确保没有一个中心节点可以被打击或者攻击，任一节点停止工作都不会影响系统整体的运作。

共识机制，通过技术来达到强安全和共识机制，无需第三方介入。

分类

广义上来看区块链大致可以分为三类：

公有区块链，全世界任何人都可读取的、任何人都能发送交易且交易能获得有效确认的、任何人都能参与其中共识过程的区块链，无需授权。全世界最大的公有链即比特币。

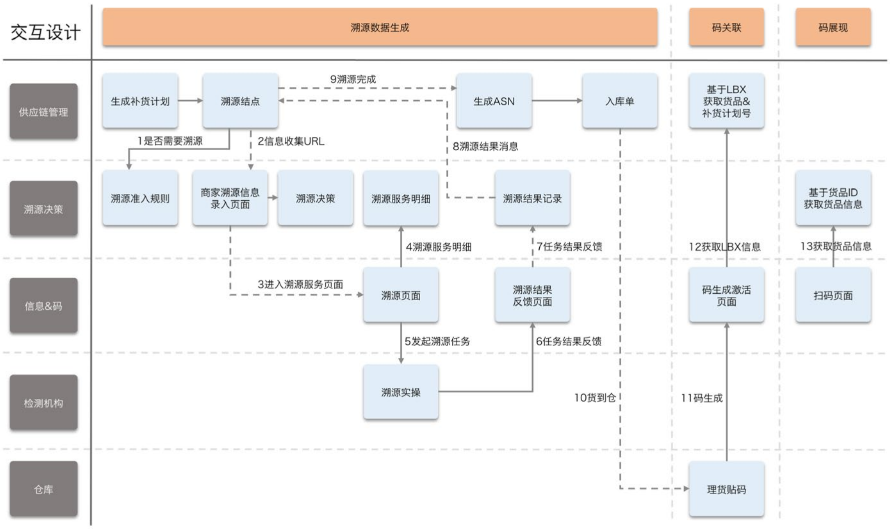
联盟区块链，共识过程受到预选节点控制的区块链，由一个联盟组织构成并对其进行管理，写入需授权接入，其他任何人可以通过该区块链开放的 API 进行限定查询。各个节点通常有与之对应的实体机构组织，各机构组织组成利益相关的联盟，共同维护区块链的健康运转。

私有区块链，写入权限仅在一个组织手里的区块链。读取权限或者对外开放，或者被任意程度地进行了限制。

针对商品溯源的特殊性，一方面我们要确保信息的干净，数据的写入必须经过授权且各方互相信任，确保写入数据的权威性。一方面要确保信息的公开透明，任何人都可以来读取，所以我们选择接入联盟链，这样一方面可以通过合作接入外部不同的质检机构达到共赢的目的，一方面确保数据的公开透明化。

技术栈

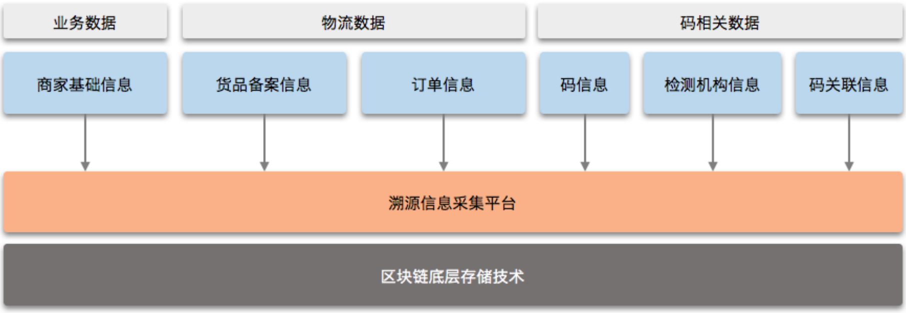
整体交互



从供应链生成补货计划到货品入库到理货贴码到最后消费者扫码获取信息，各个环节都有强有力的把控。主要思想是卡住供应链的补货流程，同时发起溯源任务，等商品溯源认证通过后，再告知商家溯源完成并继续走补货流程。

信息上链

本次天猫国际商品溯源的亮点在于成功的将区块链技术应用到溯源实操中，整个溯源流程公开透明可信赖。



一个国际货品在整个溯源流程中主要包括以下几层信息：

货品基础信息，如条形码、贸易国、生产工厂图片、工厂认证证书、成分含量图片等。

货品海关备案信息，如原产国等。

质检机构检测信息，如监装、验厂、流通、成分含量检验等。

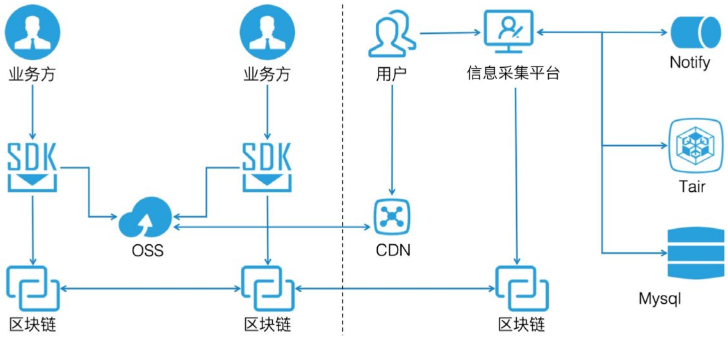
二维码关联信息。

这些信息足以确保货品的“正”与“好”，然而传统的存储都是中心化的存储，存储结果可能会被人为修改而失去公信力，溯源最重要的就是结果的真实与可信赖，项目组在如何给予消费者一个公开透明可信赖的溯源数据上做了大量的调研，最后决定引入区块链来提供强力的正品心智保障。

本次溯源项目利用蚂蚁金服的区块链数据存证能力，成功的将溯源信息放在链上，用区块链来“链接”我们和消费者。

区块链数据存证方案如下所示：

区块链数据存储方案



目前的区块链是一条**联盟链**，支持外部机构的接入，也支持商家和各类生产厂商节点的接入，以实现信息的互换和合作共赢！

平台系统设计

天猫国际建设了一整套的溯源平台系统，来联动供应链侧以共同来完成整个溯源流程。

整套系统为三层设计模型。

底层为基础的数据模型设计

基础模型，包括货品模型、工厂模型、商家模型、统计信息等，通过结构化的数据设计，将基础的底层数据构建起来，提供数据支持，数据来源有商品中心和云梯 ODPS 统计，部分货品信息来自于商家自行录入。

溯源模型，包括决策模型、溯源结果、质检模型等，由于溯源的决策规则及质检规则可能会变化，需要确保溯源模型的可扩展性。

基础配置，包括溯源准入规则配置，决策配置、白名单配置等，例如抽检的命中概率可以随时配置以适应不同时期的运营需要。

中间层是我们的业务层

业务决策，这是我们最重要的能力建设，包括几个部分，品类准入规则用来判断哪些货品需要做溯源。白名单用于某些货品或商家特殊情况跳过溯源设置。决策引擎负责判断货品命中四项抽检中的哪几项，例如验厂规则要求，以货品维度看，若无验厂报告或验厂报告已过期的，一律必须验厂。

区块链，负责数据的上下行及数据查证，具体上链方案前面已详述。

基础接入层，主要负责与外部系统之间的信息交互，登录验证，数据上链等。天猫国际平台更像是一个协调者的角色，做完业务决策后，将具体的溯源任务分发给外部质检机构，外部质检机构将结果反馈给天猫国际，国际根据自己的规则再将结果沉淀并反馈给供应链侧，以通知供应链溯源服务已经走完，可以下发 ASN。

最上层是我们的交互层

WEB 服务，包括商品溯源基础信息录入平台及小二溯源配置平台，商家可以通过基础信息录入平台录入货品的基础信息如条形码、原产地、工厂图片、成分含量图片、工厂认证图片等，供质检使用及最后传达给消费者。小二溯源配置后台负责配置溯源参数，如验厂命中概率、流通命中概率、品质退款率大于多少必须做抽检等，以免每次业务决策改变必须通过发布实现。

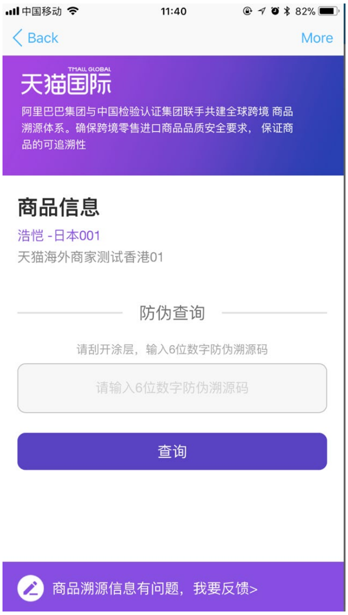
HSF 服务，外化溯源决策结果给供应链以使供应链判断是否需要溯源或者贴码，

通过 hsf 接口与其他系统间交互。

扫码演示

双十一上线后，消费者已可以收到贴有天猫国际防伪溯源码的商品，如下图所示：

打开手机扫码后页面如下图所示：



为了保证数据的安全性，二维码上有一层暗码，刮开后输入，结果如第三张图所示：

往下滑动即可看到货品的基础信息及详细的溯源信息如验厂结果、品质抽检结果等。

为了确保码不会被重复利用，当多次扫码时，会有强提示如下图所示：

展望

全球溯源计划背后离不开各国政府的支持，也是国内企业全面走向全球化的体现。目前区块链技术尚处于最初的爆发期，落地案例频现，需要人们更加深入细致的探索。而在全社会分梯次分层次迈进消费升级时代的大背景下，消费者对于消费品质的要求必将更加凸显。我们相信，将区块链的不可篡改性和可追溯性应用到商品安全上，必将进一步提高平台的正品心智，提升购物者的购物体验，打破信息壁垒，将整个供应链体系变得更加透明开放。更进一步的，我们希望借助于此，协助建立消费者对跨境零售商品溯源认知体系，以服务支撑、系统共建、信息共享为合作方式，给消费者及监管部门提供更为全面更为准确的全球商家与商品溯源信息。

直击 Weex 在优酷双 11 猫晚直播的应用

凯冯

阿里妹导读：天猫双十一已经成为被大众普遍接受的文化符号，而猫晚则是连接线下线上的重要节点。2017 年天猫晚会的前台直播任务被交给了优酷来承担。优酷直播，优酷主客团队，优酷架构组等多方组成了联合项目组，合力承担优酷双十一猫晚直播的开发任务。

缘起

虽然优酷直播在线上已经有业务稳定运行，但是我们还是遇到了大量问题需要解决。

除开直播晚会现场这个最重要的功能之外，晚会项目组还规划了点赞 / 分享有礼，竞猜，开宝箱，红包雨等五花八门的互动玩法，需要在原有的 Native 直播间上增加大量的功能。

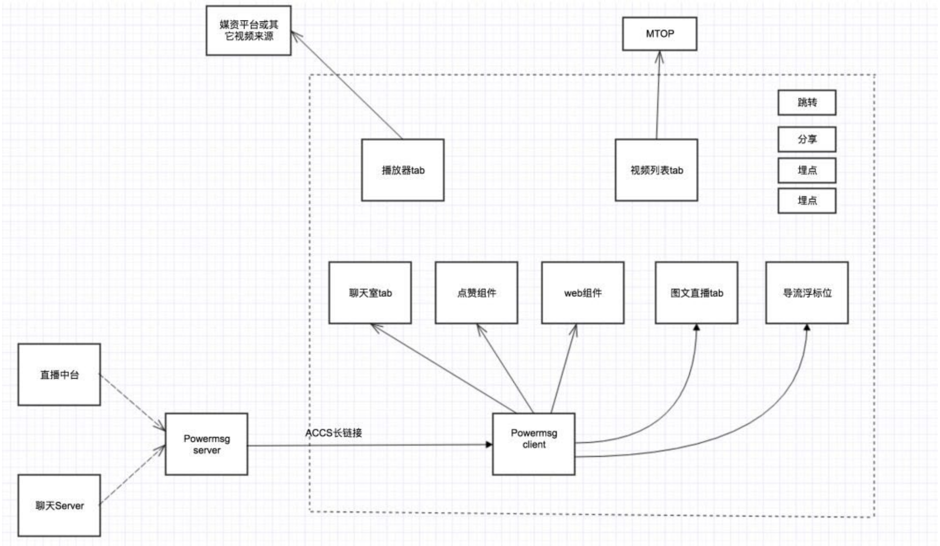
更加雪上加霜的是，晚会的招商并未结束，对直播会场的需求一直在变化。特别是直播会场换肤的需求可能导致会场的框架结构发生大幅度变化，用老的 Native 直播间应对这种需求变化比较困难。

所以我们决定用 Weex 重写一个新的优酷直播间，全面还原当前的 Native 直播间的业务，同时添加对各种双十一互动玩法的支撑。老的 Naitve 直播间仅仅作为降级的备用直播间。

组件封装

在开始正式工作之前，我们的第一件事情是划分功能模块，确定哪些模块用 Weex sdk 或者 aliweex 自带的原生组件实现 , 哪些模块需要把现有的 Native 代码封装为一个 Weex Component 供 Weex 业务代码引用。

上图即为双十一优酷直播间的最终形态，其大致的技术架构图和功能如下图所示。



经过评估，我们做出如下模块划分：

- 会场框架 : 由 Weex 业务代码搭建
- 回看列表 tab: 是一个包含图文的视频列表，由 Weex 原生实现
- 聊天 tab: 包含了大量聊天气泡动效和复杂的业务逻辑，将原有的聊天室 Native 代码封装为 Weex Component
- 图文直播 tab：包含了投放图片，文字，视频，商品链接等复杂的逻辑，将原有的 Native 代码封装为 Weex Component
- 播放器组件：将原有的优酷直播播放器 Native 代码封装为 Weex Component
- 点赞标签：这个组件被点击或者收到服务端推送的互动消息时会飘出大量的动画，我们的选择是将原有的 Native 代码封装为 Weex component
- 自定义 Tab: 这个组件我们自行封装了一个 webview 的 Component，包含多种功能

1) 利用百川能力加载淘宝商品，实现边看边买

2) 加载互动 h5 页面，实现红包雨，竞猜等功能

3) 加载阿里星球投放的 H5 页面等

其余的跳转，分享，监控埋点等功能模块封装为 Weex Module 供 Weex 业务代码使用

在功能模块拆分完毕之后，整个直播间已经完全成为组件化，模块化的，可以随意地组合，分解，定制。

无论未来的会场设计如何变化，只要编写新的会场皮肤，根据需要嵌入各种自定义标签，就能组合出需要的直播会场。

互动玩法支撑

运营同学在优酷直播中台的投放系统编辑主持人话题，换肤，红包雨等互动消息后，具体的投放信息会被服务端填入互动玩法动态模板，通过 PowerMsg 通道下发到端上；当端上收到此类互动消息之后，直接通过 globalEvent 发给 Weex 直播会场，会场解析收到的字段后，根据指令显示主持人话题，拉起宝箱，红包雨页面等。

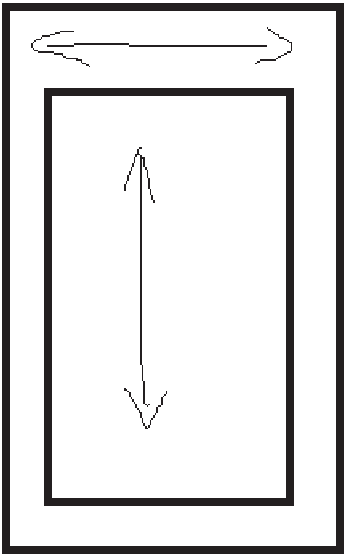
降级

与一般情况下的 Weex 页面降级到 H5 页面不同，我们的降级逻辑是发生加载 / 跳转 / 运行异常时，从 Weex 直播间跳转到 Native 直播间；所以我们没有沿用 Weex 常见的降级逻辑，而是另外实现了一套 downgrade 逻辑。

遇到的困难及其解决办法

手势冲突

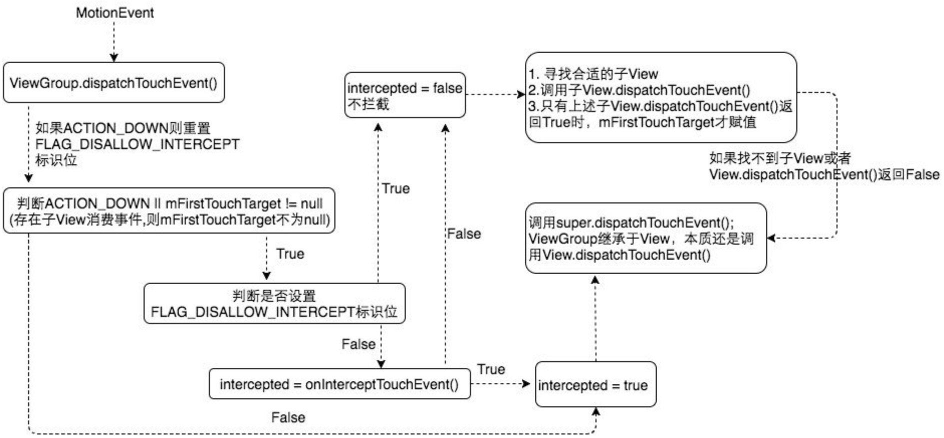
在开发直播会场的过程中，我们遇到了 tabbar 框架和自定义 Weex Component 手势冲突的问题。



我们希望上下滑动的手势被当前 tab 接收，可以在当前列表中上下滑动

我们希望左右滑动的手势被 tabbar 接收，可以在不同的 tab 之间切换

但是结合使用 tabbar 和自定义 Weex Component，要么上下左右手势全部被当前 tab 吃掉，导致无法在 tab 之间切换。要么上下左右手势都被 tabbar 吃掉，导致当前 tab 无法上下滑动



上图是 Android View 点击事件分发的简化逻辑

父 View 与子 View 同时可以滑动时就会产生滑动冲突，常见套路的一种是在父 View 的 onInterceptTouchEvent() 做拦截：

```
public boolean onInterceptTouchEvent(MotionEvent event){
    boolean intercepted = false;
    switch(event.getActionMasked){
        case MotionEvent.ACTION_DOWN:
            intercepted = false;
            break;
        case MotionEvent.ACTION_MOVE:
            if( 父控件拦截 ){
                intercepted = true;
            }else{
                intercepted =false;
            }
            break;
        ...
    }
    ...
    return intercepted;
  }
```

最后我们的解法是重写了一个自定义 Div 标签，将自定义 Weex Component 嵌入自定义 Div 标签中，自定义 Div 标签吃掉上下滑动手势并传给自定义 Weex Component，将左右滑动手势抛出让身为父 View 的 tabbar 处理，这样就解决了这个手势冲突问题。

然而，直播会场的 Weex 代码使用了 ExpressionBinding 来优化滑动性能，新的解法导致 ExpressionBinding 失效了。

为了解决这个问题，我们又重写了 Weex 的 WXGesture 手势识别代码，覆盖掉 WXComponent 自带的默认手势识别代码，使得 ExpressionBinding 重新生效。

ExpressionBinding 的执行顺序如下：

```
touchstart -> panstart -> ExpressionBinding panstart ->
ExpressionBinding panmove -> ExpressionBinding panend ->
touchend -> panend
```

其核心概念是检测出 panstart 事件，然后执行 js 层预先传下来的“手势处理逻辑”，而不是将识别出的手势传给 js 层处理。

我们对自定义 Div 标签和自定义 Weex Component 的修改使得 Expression-Binding 识别不到 panstart 事件了。所以我们重写的 WXGesture 会在合适地方给自己所在的 WXComponent 发出：

```
WXGestureType.HighLevelGesture.HORIZONTALPAN
```

或者：

```
WXGestureType.HighLevelGesture.VERTICALPAN
```

事件，人为地触发 ExpressionBinding 的识别代码执行，最终使得 Expres-sionBinding 可以与自定义 Div 标签和自定义的 Weex Component 协同工作。

转屏体验优化

之前优酷直播页面的转屏是直接将 Activity 转过来，然后让视频撑满屏幕；若要恢复竖屏则把 Activity 再转回来，恢复 vieoView 为原始大小，让其余的布局显示出来。

由于新的 Weex 直播会场复杂度大大提高，切换横竖屏的体验变得很糟糕，每次切换之后画面要黑屏一会儿才能把布局重新显示出来。

经过多番尝试，我们采用如下的解法来提高体验：竖屏转横屏时，先记录下 vid-eoView 的各种布局参数，然后将 videoView 从它的父 View 中取出，直接 attach 到当前 Window 的 decorView 上。横屏转回竖屏时，将 videoView 从 decorView 中取出，add 到旧的父 View 中，然后重设各种布局参数。

经过这个优化之后，转屏体验被大大提升了；即使在几年前的低端机器上，也能很快速地完成横竖屏切换动作。

视频圆角

如图所示，双十一直播会场的直播视频是嵌入到一个天猫电视机荧幕内的。实现方式是在自定义 video 标签之上覆盖一个天猫电视机图片，使得视频只从带圆角的框中露出。

这个实现方式在 iOS 端是没问题的，在 Android 上却失效了；视频和图片的层级存在问题，在 Android 上视频的四个边角仍然会透出来，视觉上非常难看。

我们的解决办法是给自定义播放器组件添加 "borderRadius" 属性，将 UED 设计稿中的电视机图片圆角值量出来，设置给自定义 video 标签，把这个值透传到 Native 层，经过 750px 转换之后，将视频 VideoView 在 Native 端直接切出合适的圆角。

最终，呈现出来的带圆角视频画面与电视机图片的圆角完美契合，看上去就像是一体的。

页面渲染速度和体验优化

懒加载

Weex 直播间是默认是竖屏的，但是也可以转屏幕进入横屏状态。最开始进入 Weex 直播间时，旧逻辑是同时初始化竖屏下的布局和横屏下的布局，耗时比较长。

我们的修改是进入直播间时只初始化竖屏下需要的组件，将横屏下需要的组件延迟加载，真正需要时才创建和初始化。

优化冗余布局

优化减少 Layout 层级，将设置 Visibility 为 gone 的冗余 View 全部删除。

此外，Weex 直播会场页也做了大量的层级 /View 精简的动作，两者结合起来最终使得加载体验有了很大的提高。

飘赞动画优化

直播间的点赞图标是一个 Native 封装的 Weex Component，当用户手动点击或者收到其他用户的点赞推送时，点在图标会飘出若干个红心或者服务端下发的其他图片。

粗略一看点赞功能实现的没有什么问题，但是在晚会预演的高并发条件下问题就暴露出来了，由于用户量太大，点赞推送消息太多，Android 端的点赞动画会变得非常卡顿。

我们的解决办法是做了一个 Orange 开关，在较低性能的机器上减少或者屏蔽飘赞动画；收到点赞消息推送后，不再立即飘出点赞动画，而是点赞数大于某个阈值，或者大于某个时间间隔才飘出动画，减少端上的渲染压力。

此外，我们注意到飘赞动画是飘出显示区域之后直接销毁，需要飘出新动画时再创建新的图片，我们有考虑引入对象池的技术预先分配一些动画帧并复用，但是由于时间关系未能发布到线上。

减轻服务端压力

1. zcache 将 Weex bundle JS，图片等静态资源通过 zcache 推送到客户端上。

（1）减轻了猫晚当天，客户端请求服务端拉取资源的压力

（2）由于待请求的资源就在本地，大大提高了页面渲染的速度

（3）提高了页面加载成功率

2. 请求合并由于某些 mtop 接口功能庞杂，Weex 和 Native 代码都要请求，

且调用时序无法确定。我们封装了一个 Mtop Weex module，兼顾 mtop 请求和 DataPool 的功能，Weex 和 Native 代码都通过这个 module 来发出 mtop 请求；当某个请求发出之后，短期内对同一个接口的重复请求会被缓存；第一个请求数据返回之后，结果会通知给所有被缓存的 mtop 请求方。通过这个小小的优化，我们将服务端 QPS 压力降低了十多万。

3. 猫晚当天，只请求最低限度的服务端接口，不再请求诸如“直播间预告订阅”之类的非必要接口。

双十一猫晚直播成果

最终，优酷的双 11 直播取得较好的成绩：

1. 助力天猫双 11，引导点击用户 655 万。
2. 猫晚直播同时在线达 209 万。

我们在直播会场的关键路径和优酷直播播放器均加上了埋点信息，根据统计信息：

性能指标	Android	iOS
Weex直播间加载时长	2.6s	2.1s
Weex直播间加载成功率	98.5%	100%
播放成功率	99.27%	99.07%
卡顿vv占比	3.69%	5.87%
消息到达率	99.9%	99.9%
互动H5加载成功率	99.9%	99.9%
互动H5渲染成功率	99.9%	99.9%
互动Weex渲染成功率	99.9%	99.9%

余温

以双十一直播会场为基础，依据老的标准直播间样式重写了一份皮肤之后，Weex 直播间已经在优酷直播业务中全量上线，老的 Native 直播间已经下线。我们为双十一 Weex 直播间开发的各种互动玩法已经落地为优酷直播的常态化运营手段。

此次双十一猫晚直播是优酷历史上采用 Weex 承接的规模最大的项目，也是阿里历史上直播在线人数最高的项目。在各部门的通力合作之下，整个晚会的直播任务运行平稳，没有出现任何意外情况。

在当前的双十一直播会场成果基础上，Weex 直播间不仅承担了目前线上全部的优酷直播业务，还会承担即将到来的跨年晚会，春节联欢晚会的直播任务，直播团队还将推出更多酷炫的互动玩法，更加流畅的直播加载体验，敬请期待。

如何把范冰冰“送”到你家？双 11 晚会“逆天”技术首次公开

沐节

阿里妹导读：2017 天猫双 11 晚会，1.4 亿人次边看晚会边玩边买，一场别开生面的“明星到你家”AR 互动活动令人印象深刻。晚会现场，范冰冰、卢靖姗瞬间穿越到了所有观众家中，6 分钟内 225 万人次邀请“明星到你家”。

今天，我们邀请到了阿里工程师沐节，为大家揭秘如何把范冰冰“送”到你家里。

背景介绍

每一年的天猫双 11 狂欢夜都是明星云集，2017 年当然也不例外，大半个娱乐圈都来了，还有好莱坞巨星、超级体育大咖等。

但是，如果明星们只是一如既往地呆在电视机里，那就太没有新意了。能不能让电视机里的明星走下舞台，甚至是进到每一个观众的家里去，和观众实现零距离互动呢?

范冰冰“瞬间移动”到你家

在这次晚会中，观众只要在手机上点一下按钮，就能把明星“请”到自己家里拍照、互动。

原来，阿里工程师首次把真人三维建模和动作捕捉技术运用到了国内的手机平台，把明星的动作甚至表情细节都进行了逼真的还原，再通过特殊加密算法将模型文件压缩在 20M 左右，同时巧妙地将 AR 技术与 VR 全景技术结合，实现了用技术将明星通过 AR 技术投射到真实场景，达到和观众面对面、多角度、零距离接触的效果。

机遇与挑战

在接到这样一个项目的时候，整个团队都是很兴奋的，可以把一种新技术带给大众，不仅是炫技，更是技术人心中的一种执念：技术改变世界。

进行技术调研后，我们有了以下顾虑：

1. AR 算法需要较好的性能表现。
2. 需要集成在手淘 & 猫客 APP 中，bundle 尺寸越小越好。
3. 在 iOS 平台上只有 iPhone 6S 及 iOS 11 的用户可以使用 ARKIT。
4. Android 端碎片化严重，机型差别巨大，有各种各样的兼容性问题。

针对上述问题，我们调研了一些成熟引擎：

1. 类似于 Unreal，Unity 这样的游戏引擎，具有成熟的开发生态，强大的能力。专业的效果需要专业的团队，简单的效果 Unity 这样的入门也不难。但是他们比较大并且是框架性质，直接生成一个独立 App 的大型引擎，不适合嵌入手机淘宝这样的应用中使用。
2. WebGL 随着浏览器的发展，纯渲染的能力和兼容性越来越强，会满足大量的需求。但是在面对高性能要求的 AR 各种算法，或者各种与渲染相关的强 Native 能力（譬如优化的直播视频流，图片插值内容等）时就会力不从心。等待原生的发展事实上是要等待标准的发展和普及，PWA，WebAssembly 等技术尚不能算完全可用。
3. Hybrid 方案通常会面临异构系统渲染带来的不同步的硬伤，并且高频通讯也通常带来糟糕的性能问题。另外，Native 的透出能力，除了 api 级别的透出，无论是资源使用的细粒度控制，或是需底层处理的渲染内容，都不是简单的一两个函数调用的抽象可以解决的问题。

考虑到 2017 猫晚的覆盖人群与广泛性，兼容手淘 / 猫客与各种机型 & 渠道是必然的，综上所述最终我们采用 Weex-Redim 架构解决，Weex-redim 在能嵌入淘系 App 的包大小的情况下，实现了多端一致性，动态性，高易用性和高性能的 3D/VR/AR 应用研发体系并具备开放使用的能力。

关于 Weex-Redim 的常见问题与思考

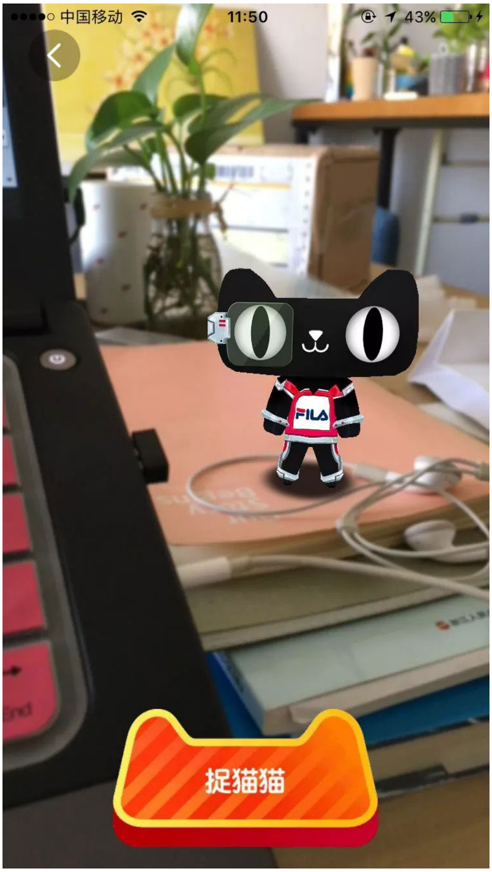
Redim 以 vuecomponents 形式调用，对于熟悉 Vue/Weex 的同学只需要了解一些图形学的概念上手几乎没有任何成本。这一点难得可贵，市面上的大部分引擎都

需要熟悉一定的图形学相关知识，开发者上手需要学习成本。

1. 如何创建一个 AR+ SLAM 的场景?

使用 <r-slam> 与 <r-ar-cursor-layout> 标签就可以创建一个 AR 场景，而 Camera/Renderer/Light 等 Redim 已经帮助处理无需关心，Tracking 也由 Redim 的 slam 算法解决，你所要做的只是在标签上指定一个算法 :type=arkit/planeasmarker/imu 就好了。

```
<r-slam type="arkit">
    <r-ar-cursor-layout>
        <slot></slot>
    </r-ar-cursor-layout>
</r-slam>
```



2. 动画支持

Redim 原生支持 Animation 动画，目前支持 translate3d 与 opacity, 一些常用的矩阵操作基本可以支持。复杂的动画支持较为有限。

3. Redim 不适合做哪些?

高质量大规模场景级渲染：对于大量内容的，灯光渲染场景，目前由于工具链的缺失，摆起来还是比较吃力。并且动态创建不确定类型和数量的节点，也是一个写起来不太方便的事儿。

高效帧级粒度的富交互：当出现实时 60fps 的计算需求并要反应在组件的属性变化时，由于 Weex 的线程转发机制，效率不高。暂时可以通过降低回调频率缓解，但是会看出来一些。

亚组件粒度的自由控制：当希望控制组件无法描述的很底层渲染特性时，如组件没有提供控制力，就无法进行自定义。除非自己扩展新组件。

Redim 有着上手快，无兼容性困扰，性能稳定，开发效率高的优势，但在大规模场景渲染上由于工具链缺乏，还存在很多不便之处，总的来说绝大部分业务场景上还是推荐同学们使用的：）

项目中的一些“坑”

1. 模型过大，50Mb 左右的模型尺寸足以让大部分用户望而却步。

解决方案比较针对性，我们通过双十一前置活动提前下发相关模型文件，晚会当晚预加载等，这种方案非常多就不一一列举了。一劳永逸还是要通过模型压缩算法啊 :)

2. 需要三种算法：

- 在 iPhone 6s+ iOS 11 平台上有苹果的硬件支持，Arkit 可以给用户最好的体验。
- 低于 iOS11 的其他系统与 Android 跑分在 75 分以上的中高端机型采用传感器 +Marker 解决。
- 剩余不在黑名单的机型采用 IMU 传感器。

3. 基于面还是特征点?

在这个问题上我们反复了很多次，基于特征点有以下问题：

- 每次启动应用它不知道设备的位置。
- 长距离和长时间的使用，误差会累计变成 drift。
- 主要体现出的问题还是定位不准确，范爷很容易就在天上了：）

基于面的检测，有以下问题：

- 面对环境不可辨认，比如纯色的墙，纯色的桌子。
- 面对室外环境。因为深度传感器有距离限制，空间大小超出限制就没有深度信息了。
- 主要体现出的问题是定位时间过长，很容易让用户失去耐心。

为了体现最好的视觉效果，我们最终还是选择了基于面的检测。

4. 不同算法对应的世界坐标不同，其单位也不同，这点需要去做统一处理。

猫晚效果图

写在最后

新零售供应链平台：零售终端技术团队持续在日常业务、台网通、线下门店业务中，基于 AR/VR 与娱乐互动、客户体验深度结合。期待前端 & 客户端人才加入我们。简历投递邮箱：mujie.zzl@alibaba-inc.com

阿里技术

扫一扫二维码图案，关注我吧

「阿里技术」微信公众号

「阿里技术」官方微博